Report No. CG-D-26-97

Improvements to the Ship Applied Fire Engineering (SAFE) Cumulative L-Curve Algorithm Using Reliability Theory

Adam T. Garland
Arthur Heinricher
Ansuman Bagchi

Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609

Brian Dolph

U.S. Coast Guard
Research and Development Center
1082 Shennecossett Road
Groton, CT 06340-6096

Final Report
October 1997

This document is available to the U.S. public through the
National Technical Information Service, Springfield, Virginia 22161

Prepared for:

U.S. Department of Transportation
United States Coast Guard
Systems, (G-S)
Washington, DC 20593-0001

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.

The contents of this report reflect the views of the Coast Guard Research & Development Center. This report does not constitute a standard, specification, or regulation.

Marc B. Mandler
Technical Director
United States Coast Guard
Research & Development Center
1082 Shennecossett Road
Groton, CT 06340-6096

Technical Report Documentation Page

1. Report No.
CG-D-26-97

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle

Improvements to the Ship Applied Fire Engineering (SAFE)
Cumulative L-Curve Algorithm Using Reliability Theory

5. Report Date
October 1997

6. Performing Organization Code
Project No: 3309.26

7. Author(s)

Adam T. Garland, Arthur Heinricher, Ansuman Bagchi, Brian Dolph

8. Performing Organization Report No.

R&DC 13/97

9. Performing Organization Name and Address

Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609

U.S. Coast Guard
Research and Development Center
1082 Shennecossett Road
Groton, CT 06340-6096

10. Work Unit No. (TRAIS)
SHRD Report No. 114

11. Contract or Grant No.
DTCG39-95-F-E00427

12. Sponsoring Agency Name and Address

U.S. Department of Transportation
United States Coast Guard
Systems, (G-S)
Washington, DC 20593-0001

13. Type of Report and Period Covered

Final Report, September 95 - July 97

14. Sponsoring Agency Code

Commandant (G-SEN-1)
U.S. Coast Guard Headquarters
Washington, DC 20593-0001

15. Supplementary Notes
The Coast Guard technical contact and COTR is Brian Dolph (860-441-2817) of the U.S. Coast Guard Research and Development Center. The project officer is CDR Kevin Jarvis (G-SEN-1), Coast Guard Headquarters.

16. Abstract

The Ship Applied Fire Engineering (SAFE) computer program evaluates the probability of limiting the spread of fire from one compartment to another throughout a vessel. The results from SAFE are expressed in terms of the frequency of expected loss of each compartment. To evaluate these frequencies, it is necessary to determine the cumulative probability of loss of each compartment in the ship as a target from all possible fire paths from all possible rooms of origin. The existing algorithm in SAFE for calculating the loss of target compartments, referred to as the "Cumulative L-Curve Algorithm," errs on the conservative side since it permits the contribution of a particular compartment to be counted more than once toward the total probability of loss for a target compartment. The approach selected to develop a mathematically valid cumulative L-Curve algorithm utilizes the mathematical tools of reliability theory, specifically for interconnected systems.

The implication of this research is that with the new algorithm, SAFE will be able to more accurately calculate the fire safety levels of all compartments. Performance-based fire protection results must accurately indicate whether compartments are under- or over- protected relative to desired objectives. The improved algorithm will help achieve this goal.

17. Key Words
SAFE
fire safety
reliability theory
ships
patrol boat
engineering methodology
computer code

18. Distribution Statement

This document is available to the U.S. public through the National Technical Information Service, Springfield, VA 22161.

19. Security Classif. (of this report)

UNCLASSIFIED

20. SECURITY CLASSIF. (of this page)

UNCLASSIFIED

21. No. of Pages

22. Price

METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (WEIGHT)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (EXACT)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in = 2.54 (exactly).

9
8
7
6
5
4
3
2
1
inches
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
cm 1

Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares(10,000 m^2)	2.5	acres	
		MASS (WEIGHT)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	0.125	cups	c
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (EXACT)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F

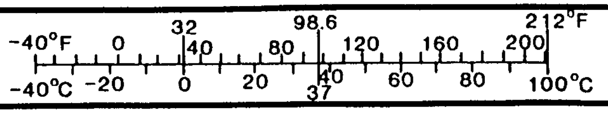


TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

Executive Summary

The purpose of this report is to present the groundwork for an improved algorithm which may be incorporated into the Ship Applied Fire Engineering (SAFE v2.2) computer program. The algorithm will more accurately calculate the "loss" of compartments to unwanted fires onboard a vessel. The currently used algorithm conservatively reports the results of a fire safety analysis conducted using the Ship Fire Safety Engineering Methodology (SFSEM) and SAFE, which means that the computed results for a particular compartment indicate a level of fire safety which is less than that which actually exists. The implication of this research is that a better, more accurate fire safety calculation method will allow more cost-efficient decisions to be made relative to required fire protection equipment and systems.

In the SAFE program, a fire is simulated as starting in a particular compartment, then the probabilities of that fire spreading throughout the ship along all possible fire paths are calculated. All compartments are not created equal relative to their frequency of fires due to different levels of fire hazard equipment within them. Therefore, a historical frequency of established burning (EB) is multiplied times the previously calculated probabilities to achieve a relative frequency of loss of compartments. These values are summed over all possible fire path combinations and the results are presented in a curve which represents the likelihood of fires being "limited" along various paths. The existing "Cumulative L-Curve Algorithm" in SAFE performs these calculations, however it errs on the conservative side by indicating that compartments are at a higher risk from unwanted fires than is actually the case. Moreover, the results for engineering spaces appear to be closer to actual conditions than non-engineering spaces. To achieve the full benefit of a performance-based fire safety analysis as evaluated by the SFSEM and SAFE, the level of safety margin (high or low) must be known. Therefore, an improved cum-L algorithm was developed.

The approach selected to develop a valid cum-L algorithm utilizes the mathematical tools of reliability theory applied to fire safety analysis. These tools come from the specific area of reliability for interconnected systems, a highly studied area of mathematics. One of the most common applications of this mathematical/statistical theory is determining the reliability of communication networks. Applied to fire safety in a ship, the *nodes* (in the network) represent individual compartments of a ship. The *edges* represent the barriers separating adjacent compartments. Using reliability theory, it is possible to determine the probability that a fire will spread from one compartment to another by examining the probabilities that each individual barrier will fail.

While an improved cum-L algorithm was successfully developed, two areas were identified which require further investigation: time dependency and barrier failure dependency. These areas are closely related and both have an impact on the calculation of the appropriate probabilities. Barrier failure dependency accounts for the two modes in which a barrier can fail (a local hot spot or a massive structural breakdown). Time dependency greatly complicates the determination of the final cumulative probabilities of loss. As fire spreads through a ship, the probabilities involved change with time. These two areas are addressed to some degree in this report, but they are subject to further investigation before they can be implemented in SAFE.

[BLANK]

1. Introduction

1.1 Ship Applied Fire Engineering

The Ship Applied Fire Engineering (SAFE) programming system, developed by the U.S. Coast Guard, evaluates the probability of spaces and barriers successfully limiting a fire on a compartment by compartment basis [1]. SAFE takes the layout of a ship, data about each compartment and barrier, and runs a probabilistic fire model on the ship data.

A run of the fire model begins with established burning (EB) at time 0 in a single compartment. The fire is allowed to progress through a set amount time by meeting certain requirements. After full room involvement (FRI) in the first compartment, the model calculates on a minute by minute basis whether the surrounding barriers will have either a thermal (T-bar) or durability (D-bar) failure. If the failure criteria for a barrier have been met, the fire is said to have spread to the adjacent compartment and the model starts established burning there. Additional information regarding SAFE is available in SAFE User Manual [1]. Other information on general fire safety engineering methods can be found in Building Firesafety Engineering Method: A Workbook [2] and Theoretical Basis of Ship Firesafety Engineering Methodology [3].

SAFE calculates the probabilities that the fire will be limited in a compartment before involving the next compartment in a fire path given that EB was established in a specific room of origin. In order to evaluate the performance of the ship relative to the firesafety objectives established for each compartment, it is necessary to express the results from SAFE in terms of the frequency of expected loss of each compartment. To evaluate these frequencies, it is necessary to determine the cumulative probability of loss of each compartment in the ship as a target from all possible fire paths from all possible rooms of origin (including itself). The notation, L, indicating limiting the fire, is relative to the allowable Magnitude of Loss (MAL) established for each compartment. For example, if a compartment has an allowable magnitude of loss rating of: EB acceptable, but not FRI (MAL = 2), the room is not considered "lost" unless fire growth in that compartment achieves FRI. Another example of a room which would be considered lost is a room with a MAL rating of 3 (FRI acceptable but not Compartment Burn Out (CBO)) which actually reaches CBO.

The existing algorithm in SAFE for calculating the loss of target compartments, referred to as the "Cumulative L-Curve Algorithm", first sums the probability of loss of a target compartment over all possible fire paths. The frequency of loss of this target compartment is then found by multiplying this probability by the frequency of EB in the room of origin. The cumulative frequency of loss of the target compartment is then found by repeating this process for all possible rooms of origin and summing the results. Finally, this process is repeated for every compartment in the ship as a possible target. This permits rapid identification of target compartments which fail to meet fire safety objectives.

Most engineering spaces are low in the ship and contain much of the hazardous materials found on board. This accounts for greater frequencies of EB in engineering spaces than other types of compartments. It is logical then that since fire tends to spread upward much easier than downward, engineering spaces are more likely to contribute to fires above than vice versa. Moreover, since the frequency of EB in these engineering spaces is an order of magnitude larger than other spaces, engineering spaces contribute heavily to the total loss in any fire path that includes them. Therefore, an engineering space may be a contributor in many fire paths that ultimately involve the same target compartment, while it may only appear once in the fire path when it is itself a target compartment. Moreover due to the relatively large contribution it provides to the cumulative loss of a target compartment, it is quite possible that the total probability of loss could exceed 1.0 for a given target compartment. If the fire paths were truly independent, the probability of loss could never exceed 1.0.

This problem was not noticed initially because the cumulative probability of loss is multiplied by the frequency of EB which is an extremely small number - even for engineering spaces. Thus the existing cum-L algorithm in SAFE errs on the conservative side since it permits the contribution of a particular compartment to be counted more than once toward the total probability of loss for a target compartment. Present results are conservative, since they indicate the ship is not as safe as it really is. Moreover, the results for engineering spaces as target compartments appear to be closer to the "truth" than non-engineering spaces. The purpose of this work effort is to develop a mathematically valid cumulative L algorithm for inclusion in SAFE.

1.2 Reliability Theory

Reliability theory is concerned with determining the probability that a system, possibly consisting of many components, will function [4]. One specific area of reliability theory is determining the reliability of interconnected systems.

Perhaps one of the most common applications of this mathematical/statistical theory is in determining the reliability of communication networks. For example, the *nodes* of a communication network might represent switch boxes and phones. The network's edges might represent the actual phone lines connecting the nodes. It is obvious that the phone companies wish for this network to be extremely reliable, meaning phone calls always go through even if an individual component may not function. More information on Reliability Theory and other applications is available in Introduction to Probability Models [4].

Another example could be the following. There is a city at a river. To cross from the north bank to the south bank there are 13 bridges connecting the banks and six islands as shown in the picture:

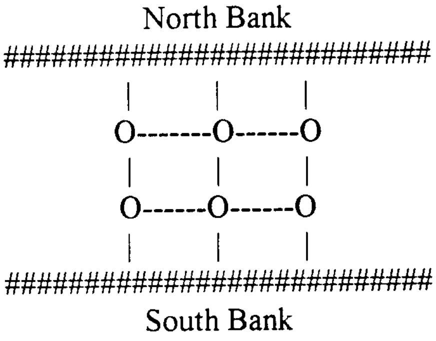


Figure 1.1: A System of Bridges Connecting Two River Banks

One night there is a storm, and each bridge has a 50% chance of collapsing. Each bridge survives independently of what happens to the other bridges. What is the probability that it is possible to cross the river from the North bank to the South bank after the storm? The solution to this example turns out to be 0.5 (the solution is described in [5]).

For this example there are $2^{13} = 8192$ possible combinations of the bridges surviving or being destroyed. Some of these combinations will contain a path from the North bank to the South bank. The brute force way of determining the probability that it is possible to cross the river is to determine which of these combinations contain a path and calculate the probability that one of these combinations occur.

In this report, it will be shown how the mathematical tools of reliability theory can be applied to fire safety. In this case, the *nodes* will represent individual compartments of a ship. The *edges* will represent the barriers separating adjacent compartments. Then it will be possible to determine the probability that a fire will spread from one compartment to another by examining the probabilities that each individual barrier will fail.

1.3 Overview of Report

This report is organized to develop the theory behind the computation of the probability of fire spread throughout a ship. In Section 2, background for determining the reliability for interconnect system will be discussed. Then several examples will be examined and analyzed. Section 3 will discuss how the above theory can be applied to the fire safety of ships.

Once the background and method has been developed, Section 4 will discuss its computer implementation. In Section 4 the method will be applied to the U.S. Coast Guard's 87' Coastal Patrol Boat. The results of this implementation will be included in the Appendices.

The last sections will discuss other computer methods, bounds on probabilities, and directions for future research. Some conclusions will be discussed in Section 8.

2. Reliability for Interconnected Systems

As mentioned, reliability theory is concerned with determining the probability that a system, possibly consisting of many components, will function. We wish to determine this probability by examining the probabilities that the individual components are functioning. For example consider the following structure:

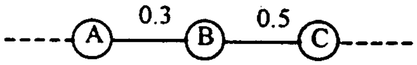


Figure 2.1: Series Structure

This is a *series* structure, where the system will function if and only if every individual component functions [4]. This structure could be a series of switchboxes in a telephone line. A call will go from switchbox A to switchbox C if and only if the phone line between A and B is working and the phone line between B and C is working. Using the probabilities that each phone line is working in the above figure, assuming independence, the probability that both are working is (0.3)(0.5) = 0.15.

Another simple example is a *parallel* structure, where the system will function if at least one of the individual components functions [4]:

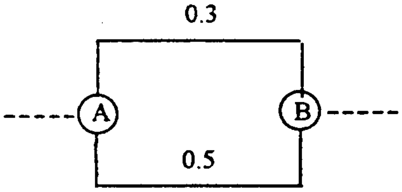


Figure 2.2: Parallel Structure

In this case, a call will go from switchbox A to switchbox B if at least one of the phone lines is working. The probability that at least one of them is working is (1-(1-0.3)(1-0.5) = (0.3)+(0.5) – (0.3)(0.5) = 0.65. To begin determining the reliability of more general structures, we must first define the structure that is to be evaluated. To do this, each structure will be represented with a graph.

2.1 Graphs

A *graph* $G=(N, E)$ consists of a set of nodes, with $|N|$=n, along with a set of edges E, with $|E|$=m [6]. We will represent each node as a single number, i, where i=1…n. We can then represent each edge by the pair of nodes they connect, (i, j). If these pairs of nodes are ordered, we have a *directed graph* and the ordered pair (i, j) represents the *directed edge* from node i to node j. If the order pair of nodes (i, j) and (j, i) represent the same edge for each existing pair (i, j) then these pairs are said to be unordered. If all of these pairs of nodes are unordered, then we have an *undirected graph*. In the analysis of fire safety, the graphs will be undirected. In either case, if the edge (i, j) exists then node i and node j are said to be adjacent.

As mentioned above, for the purpose of fire safety analysis, a graph can be used to represent such structures as buildings and ships. The nodes will represent rooms in the structure and the edges will represent the barriers separating adjacent rooms. Consider the simple graph shown in Figure 2.3:

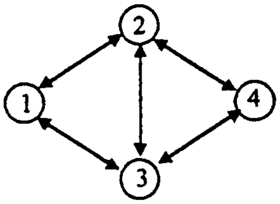


Figure 2.3: Simple Graph

Here we have an undirected graph *H=(N, E)*, with node set *N*={1, 2, 3, 4}, and with edge set *E*={ (1,2) (1,3) (2,3) (2,4) (3,4)}. To help simplify the notation each edge will be assigned a single number. To assign these indices lexicographical ordering is used. Order the edges by the first index and then by the second. For example: (1,2) < (1,3) < (2,1) < (2,3). Once ordered, we can assign each edge a number i in order. For the graph above, edge 1 is (1,2), edge 2 is (1,3), edge 3 is (2,3), edge 4 is (2,4), and edge 5 is (3,4). The graph *H* now has the following representation:

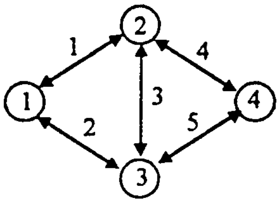


Figure 2.4: Labeled Simple Graph

Any graph (directed or undirected) can be identified with an adjacency matrix. The adjacency matrix, A, for a given graph, *G=(N, E)* is an n x n matrix, with entries:

$$A_{i,j}=\begin{cases}1, & \textit{if node i and j are adjacent}\\ 0, & \textit{otherwise}\end{cases}$$

Eq. (2.1)

The adjacency matrix for the graph H is:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Eq. (2.2)

Notice that since the graph H is an undirected graph, its adjacency matrix is symmetric. This is true for all undirected graphs. On the other hand, directed graphs do not necessarily but could have symmetric adjacency matrices. This property along with other properties and applications to adjacency matrices can be found in Graph Theory with Applications [6].

2.2 State and Probability Vectors

Having defined the graph, it is now possible to define the following two useful vectors. For each edge i, define:

$$x_i = \begin{cases} 1, \textit{ if the fire has crossed the ith edge / barrier} \\ 0, \textit{ if the fire has not crossed the ith edge / barrier} \end{cases}$$

Eq. (2.3)

Then $\mathbf{x}=(x_1,\dots,x_n)$ is known as a *state vector* [4].

The next vector that will be constructed, **p**, is called a *probability vector* [4]. Define p_i in the following manner:

$$p_i = P\{x_i = 1\} = 1 - P\{x_i = 0\}$$

Eq. (2.4)

So p_i is the probability that a fire will cross the i^{th} edge. Since each of the p_i's is a probability, it will have a value between 0 and 1.

2.3 Minimal Path Sets

We are interested in paths connecting a specific starting and ending node. The *starting node* is simply the node where the fire has started (Room of Origin). The *target node* is the node of concern (where the fire may end). We wish to determine the probability that a fire will be limited from the starting node to the target node. For example the starting node in a ship might be the engine room. The target node might then be some living quarters. The probability that a fire will spread from the starting node to the target node should be small because a fire in the living quarters could claim many lives. The methodology does not address smoke yet, but should since smoke affects life safety. To determine this probability we must find all possible paths along which the fire could spread between these nodes. The state vectors does this.

A state vector, **x**, will represent a path if it connects the starting node to the target node. In other words, the fire has spread from the starting node to the target node. Looking at Figure 2.4, with node 1 the starting node and node 4 the target node, the state vector **x**=(1,0,0,1,0) is a path because node 1 is connected to node 4 by edge 1 and edge 4. We will write this path as the set of edges it contains, {1, 4}.

Since each path represents the spread of fire, only "simple" paths are considered. By definition, a path is called a *simple* or *minimal path,* if each of the edges in the path is crossed only once and each node in the path is visited only once [6]. The set of all minimal paths of a graph *G=(N,E)* is called the *minimal path set*. Our minimal path set for Figure 2.4 is {1,4}, {2,5}, {1,3,5}, and {2,3,4} denoted as Q_1, Q_2, Q_3, and Q_4 respectively. For the sake of simplicity, we will call minimal paths, paths.

2.4 Reliability Functions

Now that all the paths along which a fire could spread from the starting node to the target node have been identified, the probability of fire spread along each of these paths must be determined. To do this we define a function of the probability vector **p**, for each path, called the *reliability function* [4]. The reliability function of a path gives the probability that the fire will spread along a series of edges in a path. Assuming independence (statistical independence) this probability is the product:

$$f_{path}(\mathbf{p}) = \prod_{i \in path} p_i$$

Eq. (2.5)

Therefore, using the minimal path set that we have constructed for Figure 2.4, we can construct the reliability function for each of the four paths, that will be named *path functions*.

$$\begin{aligned}
f_{Q_1}(\mathbf{p}) &= p_1 p_4 \\
f_{Q_2}(\mathbf{p}) &= p_2 p_5 \\
f_{Q_3}(\mathbf{p}) &= p_1 p_3 p_5 \\
f_{Q_4}(\mathbf{p}) &= p_2 p_3 p_4
\end{aligned}$$

Eq. (2.6)

Now that the individual path functions have been determined, the next step is to combine them in order to determine the probability that a fire will spread from the starting node to the target node along any of the paths.

2.5 The Algebraic Operations ⊕ and ⊗

In order to combine all the path reliability functions, we will define two algebraic operations ⊕ and ⊗ applied to polynomials [7]. The motivation for defining the operation ⊗ comes from considering how a fire spreads. In order to determine the probability that two paths occur together, their path functions need to be multiplied. If two paths do not contain common edges, then the reliability function for the combined paths is the product of the two path functions. However, if the two paths do contain one or more common edges some adjustments must be made. If we simply multiply the two functions then we will have "double counted" the probability that the fire will spread across the common edges. Therefore, in the case of common edges we just want to multiply the probability of each of these edges only once. More generally, let

$f(path_i) = p_{i_1} p_{i_2} ... p_{i_r}$ and $f(path_j) = p_{j_1} p_{j_2} ... p_{j_s}$ then define the operation ⊗ on $f(path_i)$ and $f(path_j)$ by:

$$path_i \otimes path_j = \prod_{k \in path_i \text{ or } path_j} p_k$$

Eq. (2.7)

Later on this operation will be applied to arbitrary polynomials. To do this, the operation can be extended to such polynomials by the law of distribution. If we think of each path as a set of edges then the ⊗ operation can be thought of as a union of two paths. For example if given the following two path functions

$$f = p_1p_2p_3$$
$$g = p_2p_3p_4$$

Then

$$f \otimes g = p_1p_2p_3p_4$$

Another example to show distributivity is given the following arbitrary polynomial

$$f = p_1p_2 + p_3$$
$$g = p_2p_3p_4$$

Then

$$f \otimes g = p_1p_2p_3p_4 + p_2p_3p_4$$

The next operation ⊕ is constructed to combine two reliability functions that represent parallel paths, paths that start from the same starting node and ends at the same target node. To do this we can apply the principle of inclusion – exclusion:

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(AB)$$

Eq. (2.8)

The probability of event A or event B happening is equal to the probability of event A occurring plus the probability of event B occurring minus the probability that both event A and event B occurr [8]. Letting the probability of event A be the reliability function f and the probability of event B be the reliability function g, the adaptation of this rule applied to polynomials becomes the following definition for ⊕.

$$f \oplus g = f + g - f \otimes g$$

Eq. (2.9)

If we have constructed a minimal path set for a given graph $G=(N, E)$ as defined in Section 1.3 and we have constructed the reliability function $f_i(\mathbf{p})$ for each path in the set then we can define the reliability polynomial for a given start and target node as:

$$R_{st}(\mathbf{p}) = \oplus \sum_i f_i(\mathbf{p}) = f_1(\mathbf{p}) \oplus f_2(\mathbf{p}) \oplus \ldots \oplus f_m(\mathbf{p})$$

Eq. (2.10)

In other words, the reliability polynomial is the 'sum' of path functions $f_i(\mathbf{p})$ taken over all paths from s to t [7]. To see that this is true we can go back to the definition ⊕. The operation ⊕ was derived from the principle of inclusion-exclusion. This can be expanded to more than two events. The same is true for multiple ⊕ operations.

From the definitions of the operations ⊕ and ⊗, the following properties can be easily verified. (See Appendix A).

$$\begin{array}{ll}
f \oplus f = f & f \otimes f = f \\
f \oplus g = g \oplus f & f \otimes g = g \otimes f \\
f \oplus (g \oplus h) = (f \oplus g) \oplus h & f \otimes (g \otimes h) = (f \otimes g) \otimes h \\
f \oplus (f \otimes g) = f & f \otimes (f \oplus g) = f \\
f \otimes (g \oplus h) = (f \otimes g) \oplus (f \otimes h) & f \oplus (g \otimes h) = (f \oplus g) \otimes (f \oplus h) \\
f \oplus 0 = f & f \otimes 0 = 0 \\
f \oplus 1 = 1 & f \otimes 1 = f
\end{array}$$

Eq. (2.11)

If *f*, *g*, and *h* are path functions then the above properties can be explained in the following manner. Notice the property $f \oplus f = f$, this states the probability of a path combined with itself would be exactly equal to itself. Combining *f* with 0, being a path with probability 0, then again it would just be the path function *f*. Conversely, combining *f* with 1, being a path that the fire <u>will</u> spread across with probability 1, then the probability that the fire will spread is 1.

Continuing with our example from Figure 2.4, the reliability polynomial will be:

$$\mathrm{R}_{1,4}(\mathbf{p}) = p_1p_4 \oplus p_2p_5 \oplus p_1p_3p_5 \oplus p_2p_3p_4$$

Eq. (2.12)

When this equation is expanded using the definitions of ⊕ and ⊗, we get the following:

$$\mathrm{R}_{1,4}(\mathbf{p}) = p_1p_4 + p_2p_5 + p_1p_3p_5 + p_2p_3p_4 - p_1p_2p_3p_4 - p_2p_3p_4p_5 \\ - p_3p_4p_5p_1 - p_4p_5p_1p_2 - p_5p_1p_2p_3 + 2\,p_1p_2p_3p_4p_5$$

Eq. (2.13)

which is the reliability polynomial for the probability that a fire will spread from node 1 to node 4. Once the probability vector **p** has been determined, this is the probability that a fire will spread from node 1 to node 4.

2.6 Determining the Reliability Functions of Simple Graphs

Before discussing how to determine the proper probability vector **p**, lets take a look at determining the fire spread probability of three simple graphs using some numerical values.

2.6.1 Four Nodes and Two Paths

The first graph we will consider has four nodes, and two paths:

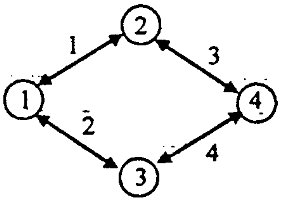


Figure 2.5: Four Node, Two Path Graph

In this graph, we have nodes 1 through 4, along with edges 1 through 4. Defining node 1 as the starting node and node 4 as the target node, then the set of paths is {1,3} and {2, 4}. The path functions for these are:

$$f_1(\mathbf{p}) = p_1 p_3$$
$$f_2(\mathbf{p}) = p_2 p_4$$

Eq. (2.14)

Applying Equation 2.10 we have:

$$\mathrm{R}_{1,4}(\mathbf{p}) = p_1 p_3 \oplus p_2 p_4 = p_1 p_3 + p_2 p_4 - p_1 p_2 p_3 p_4$$

Eq. (2.15)

Now assign some numerical values for the edge probabilities. For example, if $p_i = (0.1)i$, then the probability that the fire will spread from node 1 to node 4 is:

$$\begin{aligned}\mathrm{R}_{1,4}(\mathbf{p}) &= p_1 p_3 \oplus p_2 p_4 \\ &= p_1 p_3 + p_2 p_4 - p_1 p_2 p_3 p_4 \\ &= (0.1)(0.3) + (0.2)(0.4) - (0.1)(0.2)(0.3)(0.4) \\ &= 0.1076\end{aligned}$$

Eq. (2.16)

Notice this is not equal to the sum of the two paths 0.0300+0.0800 = 0.1100. The formula accounts for the probability that the fire is spreading across both paths. Also notice that, since the paths do not contain any common edges, the $\mathrm{R}_{s,t}$ value is equal to the sum of the two path probabilities minus the product of the two path probabilities.

2.6.2 Five Nodes and Two Paths

Consider next a five node, two path graph:

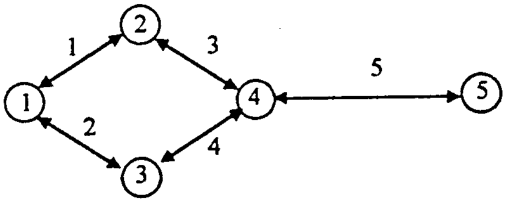


Figure 2.6: Five Node, Two Path Graph

This graph is the same as Figure 2.5 but with a fifth node and edge added at node 4. The path set for this graph, with node 1 as the starting and node 5 as the target node, is {1, 3, 5} and {2, 4, 5}. The path function for each of these is:

$$f_1(\mathbf{p}) = p_1p_3p_5$$
$$f_2(\mathbf{p}) = p_2p_4p_5$$

Eq. (2.17)

Again, applying Equation 2.10, we get the reliability polynomial:

$$R_{1,5}(\mathbf{p}) = p_1\,p_3\,p_5 \oplus p_2\,p_4\,p_5 = p_1\,p_3\,p_5 + p_2\,p_4\,p_5 - p_1\,p_2\,p_3\,p_4\,p_5$$

Eq. (2.18)

Now, assign numerical values for the edge probabilities. As before: let $p_i = (0.1)i$. Then the probability that the fire will spread from node 1 to node 5 is:

$$\begin{aligned} R_{1,5}(\mathbf{p}) &= p_1\,p_3\,p_5 \oplus p_2\,p_4\,p_5 \\ &= p_1\,p_3\,p_5 + p_2\,p_4\,p_5 - p_1\,p_2\,p_3\,p_4\,p_5 \\ &= (0.1)(0.3)(0.5) + (0.2)(0.4)(0.5) - (0.1)(0.2)(0.3)(0.4)(0.5) \\ &= 0.0538 \end{aligned}$$

Eq. (2.19)

Notice again this is not equal to the sum of the two paths 0.0150 + 0.0400 = 0.0550. Also notice, because the paths do contain commons edges, the $R_{s,t}$ is not equal to the sum of the two path probabilities minus the product of the path probabilities.

2.6.3 Four Nodes and Four Paths

The last example we consider is the example used in the beginning of section 2. We have already found the reliability polynomial for this graph but assign some numerical values to the probabilities as in the above examples. Recall that the graph looks like:

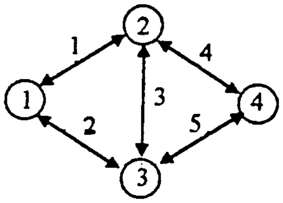


Figure 2.7: Four Node, Four Path Graph

Also recall that the reliability polynomial with node 1 as the starting node and node 4 as the target is:

$$R_{1,4}(\mathbf{p}) = p_1p_4 + p_2p_5 + p_1p_3p_5 + p_2p_3p_4 - p_1p_2p_3\,p_4 - p_2p_3p_4\,p_5$$
$$- p_3p_4p_5\,p_1 - p_4p_5p_1\,p_2 - p_5p_1p_2\,p_3 + 2p_1p_2p_3p_4\,p_5$$

Eq. (2.20)

Substituting in the numerical value of the probability vector **p**; $p_i = (0.1)i$, we get the following answer:

$$\begin{aligned} R_{1,4}(\mathbf{p}) = {} & (0.1)(0.4) + (0.2)(0.5) + (0.1)(0.3)(0.5) + (0.2)(0.3)(0.4) \\ & - (0.1)(0.2)(0.3)(0.4) - (0.2)(0.3)(0.4)(0.5) \\ & - (0.3)(0.4)(0.5)(0.1) - (0.4)(0.5)(0.1)(0.2) \\ & - (0.5)(0.1)(0.2)(0.3) + 2(0.1)(0.2)(0.3)(0.4)(0.5) \\ & = 0.1540 \end{aligned}$$

Eq. (2.21)

In this section a method to determine the reliability function for two nodes in a graph was developed. To determine the reliability function the ⊕ sum of the paths between the nodes was computed. In the examples it can been seen that the number of terms in the reliability function grows quickly with the number of paths. Now that the method has been established, how this method can be applied to ships will be discussed in the following section.

3. Application to U.S. Coast Guard Cutters

The "reliability" of a ship can be thought of as its resistance to fire spread. If a fire starts in a single room and is contained and or extinguished without spreading to any other rooms, the ship is reliable as a fire safety system. On the other hand if a fire starts in a single room and spreads to several other rooms, the ship is less reliable. Therefore by determining the probabilities of fires spreading from one room to another we can measure the reliability of a ship's fire safety. If there are several high probabilities that a fire with spread from one room to another, then adjustment can be made to help lower these probabilities and make the ship more reliable. To start determining these probabilities we must first represent a ship's layout by a graph.

3.1 Representing a Ship with a Graph

If we are given the layout of a ship we can easily represent this ship with a graph in the following manner.

1. Represent each room or compartment in the ship as a single node.
2. If any two rooms are adjacent or separated by a barrier, connect the corresponding nodes with a single edge.
3. Assign a probability to each edge. Since each edge in the graph represents a single barrier we can assign to it the probability that the barrier will fail (and the fire will spread across it).

To illustrate this we will apply this method to an early design version of the USCG Coastal Patrol Boat (CPB). The layout of the CPB can be seen in Appendix D. Also, additional information can be found in the report: Fire Safety Analysis of the Coastal Patrol Boat [9]. After applying the first two steps the adjacency matrix representing the CPB's graph is

$$
\underline{A} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0
\end{bmatrix}
$$

Eq. (3.1)

The CPB has 26 separate compartments over three different decks. Therefore the adjacency matrix is a 26 x 26 symmetric matrix. There are a total of 69 barriers represented by 69 edges. Each of these edges is represented by two entries in the adjacency matrix, i.e. the barrier that separates the i^{th} and j^{th} barrier is represented by the (i,j) and (j,i) entry.

The following chart shows which index represents each compartment in the ship

Node	Compartment
1	2-0-0-W
2	2-2-0-L
3	2-5-0-L
4	2-5-1-L
5	2-5-2-L
6	2-7-1-L
7	2-7-2-L
8	2-9-1-Q
9	2-9-2-L
10	2-13-0-Q
11	2-14-0-E
12	2-22-1-Q
13	2-22-2-Q
14	1-7-0-Q
15	1-8-0-Q
16	1-8-1-L
17	1-8-2-L
18	1-11-0-L
19	1-11-1-L
20	1-11-2-L
21	1-12-0-Q
22	1-12-1-L
23	1-12-3-Q
24	1-15-1-T
25	1-15-2-T
26	01-11-0-C

Figure 3.1: Node to Compartment Conversion Chart

The notation for each compartment above is used in [9]. The first of the three numbers are used to show which deck the compartment is on. The second and third numbers are used to show where the compartment is located on its respective deck. The letter is used to show what type of compartment it is, i.e. L stands for a living compartment, where E stands for an engine room. Now that the adjacency matrix has been constructed let's take a look at assigning the probabilities to each edge.

3.2 Assigning a Probability to Each Edge

Determining the probability that a fire will spread from one compartment to another involves a new reliability graph. Once established burning (EB) has occurred in a compartment, there are three steps that can be used to stop the fire from fully involving the compartment:

1. I – The fire self terminates
2. A – Automatic: The fire is automatically suppressed
3. M – Manual: The fire is manually suppressed

Each of these events also has a failure complement, $\bar{I}$, $\bar{A}$, and $\bar{M}$ (read as I-bar, A-bar, and M-bar respectively.) All of these steps act independently of each other but in the sequence above. If all three of these steps fail the fire is said to have reached the state L-bar. If any of these steps extinguish the fire then the state L has been reach. In order to calculate L-bar consider the following network.

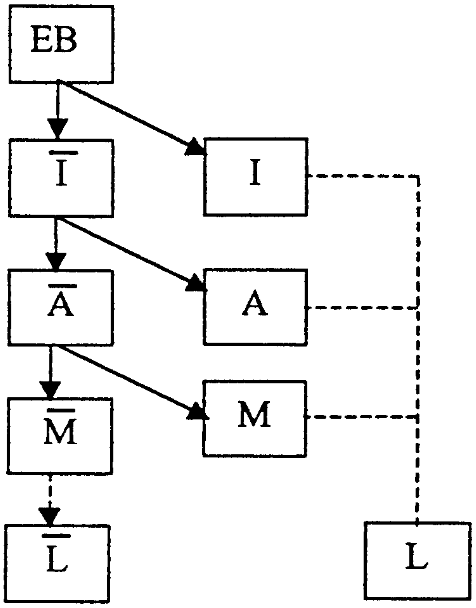


Figure 3.2: L-bar Network

Given the above graph, we can easily find the reliability function using EB as the starting node and L-bar as the target node. There is only one path from EB to L-bar and it has a series structure. Therefore its path function is the product of the individual edge probabilities involved in the path from EB to L-bar,

$$R_{EB,I\text{-bar}}(p) = p_{I\text{-bar}}p_{A\text{-bar}}p_{M\text{-bar}}.$$

Eq. 3.2

In other words the probability that a fire will not be extinguished in a compartment is $p_{I\text{-bar}}p_{A\text{-bar}}p_{M\text{-bar}}$. Additional information on calculating L-bar probabilities is available in Building Firesafety Engineering Method: A Workbook [2] and the Theoretical Basis of the Ship Fire Safety Engineering Methodology [3].

Once a compartment has reached the state L-bar, the fire may still be contained in it by its surrounding barriers. In order to spread to an adjacent compartment the fire must go through at least one of the barriers separating the fully involved room from it's adjacent rooms. The probability of this occurring depends on the material of each barrier.

Throughout the ship there are different types of materials used to construct barriers. For each of these materials we can calculate two probabilities, the probability of a T-bar failure (thermal failure) and the probability of a D-bar failure (durability failure). We can think of each of these possibilities as independent parallel paths or events. Therefore the probability of a barrier failure is:

$$p_{T\text{-bar}} \oplus p_{D\text{-bar}} = p_{T\text{-bar}} + p_{D\text{-bar}} - p_{T\text{-bar}}p_{D\text{-bar}}$$

Eq. (3.3)

Now lets look at a single edge connecting two nodes and determine its probability. At first this small section of the graph would be:

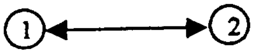


Figure 3.3: Two Nodes, One Edge

Incorporating the ideas as described above, a more detailed representation is:

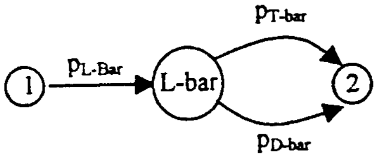


Figure 3.4: Detailed Representation of Two Nodes, One Edge

Notice that the new graph has directed edges, meaning the fire cannot spread from L-bar to node 1. Looking at the above figure one might think that the probability between any two nodes is $(p_{L\text{-bar}})(p_{T\text{-bar}} \oplus p_{D\text{-bar}})$ using the correct L-bar, T-bar, and D-bar calculations. This however is not exactly true. To illustrate this, apply above method to the following section of graph.

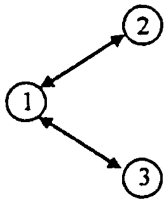


Figure 3.5: Three Nodes, Two Edges

This section of graph now becomes:

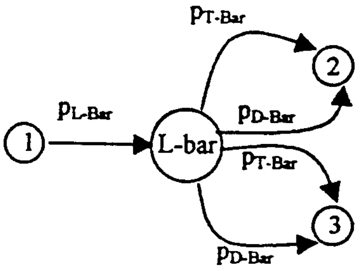


Figure 3.6: Detailed Representation of Three Nodes, Two Edges

The above figure shows that the edge, which connects node 1 to node 2, and the edge that connects node 1 to node 3 in Figure 3.5 are no longer independent. In order for the fire to spread from node 1 to any other node it must first achieve L-bar.

There are two methods to incorporate the necessary changes and to calculate the proper probabilities. The first involves a transformation of the original adjacency matrix to incorporate the imaginary "L-bar" nodes. The second method involves multiplying each term in the final reliability polynomial with the proper L-bar probabilities.

Without applying either of the above two methods the best estimate of the probability for each arc is $(p_{L\text{-bar}})(p_{T\text{-bar}} \oplus p_{D\text{-bar}})$, for the corresponding node and barrier. Therefore these are the values that will be used throughout the rest of this report, and the existences of dependence will be disregarded. The adjacency matrix is symmetric, but the probability matrix is not necessarily symmetric since the probability of fire spread through a barrier may be different depending on the direction of fire spread. The implementation of the above two methods will be discussed in Section 7: Implementation of Time and L-bar Dependencies.

4. Computer Implementation

Now that a method for computing a ship's fire spread probability has been derived, its computer implementation can be developed. From what has been discussed above there are four main steps in computing a ship's fire spread probability. The first step is representing the ship with the proper adjacency matrix and reading it in as input. The second step is to find all paths from the starting node to the target node. The third step is $\oplus$ summing these paths to determine the reliability polynomial, $R_{s,t}(\mathbf{p})$. The final step is to calculate the numerical value of $R_{s,t}(\mathbf{p})$ using the ship's probability vector $\mathbf{p}$.

4.1 Data Input

All the information needed at this point is to be read in from two files. The first file is adj.txt, which contains the adjacency matrix for the ship. The second file will be probs.txt, which contains the probability matrix with the probabilities for each edge. Another computation that is done in the input is the numbering of the edges, as described in Section 2.1.

- Definition of new variables used in this algorithm:

A[][]	– The adjacency matrix
P[][]	– The probability matrix
Edges[][]	– Stores the edge number k for each (i,j)
count	– Used to index the edges

- Algorithm 1: Input

```
Read in the size of the adjacency matrix, n
For i = 1 to n
    For j = 1 to n
        Read in A[i][j]
        Read in P[i][j]
        If A[i][j] $\neq$ 0
            count = count + 1
            Edges[i][j] = count
```

Continuing with the example from Section 1, the adjacency matrix read in from the file adj.txt is

$$A=\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Eq. (4.1)

The probability matrix is

$$P=\begin{bmatrix} 0 & p_1 & p_2 & 0 \\ p_1 & 0 & p_3 & p_4 \\ p_2 & p_3 & 0 & p_5 \\ 0 & p_4 & p_5 & 0 \end{bmatrix}$$

Eq. (4.2)

where the values p_i are to be determined later.

4.2 Finding all Paths from the Starting Node to the Target Node

The second step in determining a ship's fire spread probability is finding all paths from a starting node, s, to a target node, t. The number of all paths between a pair of nodes, however, can be exceptionally high. To alleviate the computational burden, the implemented algorithm allows the user to prescribe an upper bound on the length of paths to be used. There are two advantages to this. First, a bound reduces the number of paths to be explored in all future computations. Second, the probability of fire propagation along long paths is expected to be very low. If these long paths are disregarded, the accuracy of the overall probabilities is good. The paths are stored in a file and the first number in the file is the total number of paths. The format used in the rest of the file paths.txt will be the following.

$$L \quad P \quad x_1 \quad x_2 \ldots x_L$$

Figure 4.1: Format for Paths.txt

where L is the length of the path, P is an indicator discussed later, and $x_1, x_2, \ldots, x_L$ is the list of nodes describing the paths. Each new line will represent a different path, so when the file is used later if the i^{1h} path is needed then one can just go to the i^{th} line of the file and extract the path. The reason for the indicator will be explained in Section 4.4: Calculating the numerical value of $R_{s,t}(\mathbf{p})$, using the ship's probability vector **p**. Suppose $s = x_0, x_1, \ldots, x_L = t$ is a path from the source node s to the target node t. We may call each x_{i-1} as the parent of x_i $(1 \leq i \leq L)$ along this path.

Using the following algorithm the paths from the node *s* to node *t* will be found and be represented as a list of nodes in the path. This representation of a path will be called a node path. Similarly each of these node paths will be translated into a list of edge connecting each consecutive pair of nodes in the node path. This representation of a path will be called an edge path.

- Algorithm 2: Finding all paths of length k or less:

```
Part 1: Creating the information array
For each node create a list of adjacent nodes called neighbors
Read in the maximum path length k, start node s, and target node t
Set the starting node as the current node
While the current node is not the last node in the array
    If the current node is not the target node and the maximum path length has not
    been reached then
        For each neighbor of the current node
            Check all parent nodes back to root,
            If the neighbor is not in this list then
                Add it to the end of the queue and mark the current node as
                its parent
    Set current node to next node in the queue
```

Part 2: Extracting the node paths from the information array

```
For each node in the queue
    If current node is equal to the target node then
        Follow the current node's parents back to the root
        Set path equal to list of nodes created by following parents
```

Part 3: Translating each node path to an edge path

```
For each node path
    For each pair of consecutive nodes in the path
        Add the edge connecting the pair of nodes to the edge path
```

To illustrate this we will use the graph in Figure 2.4. The starting node is node 1 and the target node is 4. The bound used on this example is 4. The first step is to find all the neighbors for each node.

Node	Neighbors
1	2, 3
2	1,3,4
3	1,2,4
4	2,3

Set Node 1 as the first node in the queue. Node 1 has neighbors 2 and 3. Since this is the first part of the queue, these nodes are not parents yet. After checking the starting node the queue will look like

Queue Position	Node	Parent Position
1	1	-
2	2	1
3	3	1

Now that the first node has been checked, the second node in the queue will be checked and the proper neighbors will be added

Queue Position	Node	Parent Position
1	1	-
2	2	1
3	3	1
4	3	2
5	4	2

The second node has now been checked and we will move onto the third and fourth node

Queue Position	Node	Parent Position
1	1	-
2	2	1
3	3	1
4	3	2
5	4	2
6	2	3
7	4	3
8	4	4

Notice that only one neighbor is added to the queue from the fourth parent node 3. Its neighbor node 1 was not added because it was the starting node. Node 2 was not added because it is the parent of the current node being checked. The neighbor, node 4 was added. Finishing the queue it will be

Queue Position	Node	Parent Position
1	1	-
2	2	1
3	3	1
4	3	2
5	4	2
6	2	3
7	4	3
8	4	4
9	4	6

Notice once the array is completely built, node 4 occurs four times. This is the same number of paths from node 1 to node 4. Now the paths can be extracted, for example the node in position 9 is the target node. Its parent is in position 6, node 2, which in turn has the parent node 3, which in turn has the parent node 1. The list of nodes is then {4, 2, 3, 1}. Ordering this list in the opposite order we get the list of nodes {1, 3, 2, 4}, which is one of the node paths in Figure 2.4.

The final step to this algorithm is to translate each node path to an edge path. Given the above node path, {1, 3, 2, 4} it's edge path would be {2, 3, 4}. Node 1 and node 3 are connected by edge 2. Node 3 and node 2 are connected by edge 3. Finally node 2 and node 4 are connected by edge 4. Notice each edge path will contain one less number in the set then its respective node path.

Doing this for each target node found in the queue and printing each path to a file at the end the paths.txt file for Figure 2.4 is

```
4
2 1 1 4
2 1 2 5
3 1 1 3 5
3 1 2 3 4
```

Figure 4.2: Paths.txt file for Figure 2.4

Examining this output we can see there are 4 paths, {1,4}, {2,5}, {1,3,5}, and {2,3,4}. Referring back to Section 2.3, we can see that this output matches the paths found earlier.

4.3 ⊕ Summing Paths from Node s to Node t

Now that all the paths from node s to node t, of a specified bound, have been found, the reliability polynomial $R_{st}(p) = \oplus\sum_{i} f_i(p)$ can be constructed. At this point there exists a file called paths.txt, which contains the list of paths. The file solution.txt will also be used to store the ⊕ sum. The format used to store the sets of edges will be the same as that used to store the paths. The main principle used to calculate this sum comes from the associative property of the operations ⊕, $f \oplus (g \oplus h) = (f \oplus g) \oplus h$.

- Algorithm 3: Calculating the ⊕ sum of a list of paths:

```
Read in the total number of paths, N
For i = 1 to N
        Read in the current path
        Print the current path to the file temp.txt
        Go to the start of solutions.txt
        For j = 1 to $2^{(i-1)}-1$
        Read in the $T^h$ solution set
                Find the union of the current path and the current solution set
                Print the union to the file temp.txt
                If both the current path and current set are positive, mark the current set as
                negative
                If both the current path and current set are negative, mark the current set
                as negative
                If only one of the current sets is positive, mark the current set as positive
        Read in all information from temp.txt and print it to the end of solution.txt
Print a marker to the end of the file solution.txt (i.e. 99999)
```

At the end of this subroutine, the solution.txt file for Figure 2.4 is

```
2 1 1 4
2 1 2 5
4 -1 1 2 4 5
3 1 1 3 5
4 -1 1 3 4 5
4 -1 1 2 3 5
5 1 1 2 3 4 5
3 1 2 3 4
4 -1 1 2 3 4
4 -1 2 3 4 5
5 1 1 2 3 4 5
5 -1 1 2 3 4 5
5 1 1 2 3 4 5
5 1 1 2 3 4 5
5 -1 1 2 3 4 5
99999
```

Figure 4.3: Solution.txt file for Figure 2.4

Examining the output, we can see that each line represents a single term in Equation 2.13. Recall that the first number indicated the number of p_i values in the term. The second number in the set indicates whether the term it represents has a positive (1) or negative (-1) value. The rest of the numbers are the set of p_i's in the term. For example, the first line, 2 1 1 4, is a set of 2 edges with positive value. The set is {1,4}, which represents p_1p_4, the first term of Equation 2.13. On the other hand the third line, 4 -1 1 2 4 5, is a set of 4 edges with a negative value. The set is {1, 2, 4, 5}, which represents $-p_1p_2p_4p_5$, the fifth term in Equation 2.13. Note that the last four lines, not including the marker 99999, represent four equal terms. But two of the terms have a positive indicator and two of them have negative indicator. Therefore these terms will cancel each other.

4.4 Calculating the Numerical Value of $R_{s,t}(p)$, Using the Correct p

Given the list of edges sets, a value can be assigned to edge and the final numerical value for the probability that a fire will spread along any path of length k or less can be found.

- Definition of new variables used in this algorithm:
  answer – $R_{s,t}(\mathbf{p})$ using the correct **p**
  total – The current total of the list of sets.
  value – The numerical value of the current set edges

- Algorithm 4: Calculating the numerical value of $R_{s,t}(\mathbf{p})$

```
Read in the first set of edges
While the current set of edges is not equal to the end of file marker
        Set value equal to $p_i p_j \ldots p_r$ where i, j, …, r is the current set of edges.
        If the current set is mark as positive then
                total = total + value
        If the current set is mark as negative then
                total = total – value
        Read in next set of edges
   Set answer equal to total
```

Let the probabilities for the graph in Figure 2.4 be

$p_1 = 0.1$, $p_2 = 0.2$, $p_3 = 0.3$, $p_4 = 0.4$, $p_5 = 0.5$

Using these probabilities set as in Section 2.6.3, the computer output for $R_{s,t}(\mathbf{p})$ is:

```
A solution has been found and is: 0.1540
```

Comparing this to the answer found in Section 2.6.3 we can see that they are equal.

4.5 Determination of the Probabilities for the Entire Ship.

In order to solve the probabilities between each and every room we can do the following. By placing a double 'for' loop around the entire process above, the calculations from every room to every other room can be made.

- Algorithm 5: Solving for the entire ship

```
Let n be the total number of compartments
for s = 1 to n
        for t = 1 to n
                Finding all paths from the starting node to the target node.
                $\oplus$ summing paths from node s to node t.
                Calculating the numerical value of $R_{s,t}(\mathbf{p})$, using the correct p.
                Answer[s][t] = answer
```

The output will be an n x n matrix where the (i,j) entry is the probability that the fire will spread from node (or compartment) i to node (or compartment) j.

5. Other Iterative Methods

In the previous section one method of implementing reliability theory with computers was discussed in detail. The algorithm evaluates only one reliability function, $R_{s,t}$, dependent on the starting node s and the target node t. If the probability for spread of fire between two different nodes were to be determined, the algorithm would need to be applied again using the new starting and target node. In this section we will take a look at two more iterative methods that can be used to solve graph reliability. Both of these methods will only be dependent on the starting node. Once a starting node has been chosen the following algorithm will solve for each and every target node in the graph. More information on each of these methods can be seen in [7]. The first thing needed to do for both of these algorithms is to derive a system of equations that will be solved.

5.1 Constructing a Set of Equations for Determining the Reliability of a Graph.

While it is possible to associate an adjacency matrix with a given graph G, it is also possible to associate a system of equations with graph G. Assume G has n nodes. For each node j $(1 \leq j \leq n)$ in the graph, a value z_j, the probability that a fire will spread from the starting node to node j, can be associated to it in the following manner. For each node j there is a finite number of adjacent nodes i. Each of these nodes has an edge e connecting it to node j. Associated with each of these edges is a probability p_e. Throughout this section the value p_e will equal $p_{i,j}$ where e is the edge connecting node i to node j.. Using this notation the value z_j is then defined as:

$$z_j = \oplus \sum_{(i,j)\in E} p_{i,j} \otimes z_i$$

Eq. (5.1)

In other words the probability that a fire will spread from the starting node to node j, z_j, is equal to the $\oplus$ sum over all adjacent nodes i, of the probability that a fire will spread to and adjacent node i, z_i, multiplied by the probability that the fire will spread from node i to node j, $p_{i,j}$ [7].

Continuing with the example used throughout Section 2, the system of equations for the graph in Figure 2.4 is:

$$\begin{aligned}
z_1 &= (p_1 \otimes z_2)(p_2 \otimes z_3)\\
z_2 &= (p_1 \otimes z_1)(p_3 \otimes z_3)(p_4 \otimes z_4)\\
z_3 &= (p_2 \otimes z_1)(p_3 \otimes z_2)(p_5 \otimes z_4)\\
z_4 &= (p_4 \otimes z_2)(p_5 \otimes z_3)
\end{aligned}$$

Eq. (5.2)

In order to choose a starting node the following can be observed. The probability that a fire will spread from starting node to the starting node is equal to 1. Therefore the solution to equation z_s, where s is the starting node, is already known to be 1. If in the above system of equation the starting node is node 1, the system of equations then becomes:

$$z_1 = 1$$
$$z_2 = (p_1 \otimes z_1)(p_3 \otimes z_3)(p_4 \otimes z_4)$$
$$z_3 = (p_2 \otimes z_1)(p_3 \otimes z_2)(p_5 \otimes z_4)$$
$$z_4 = (p_4 \otimes z_2)(p_5 \otimes z_3)$$

Eq. (5.3)

As mention earlier the equation z_j was to be defined as the reliability polynomial $R_{s,j}(\mathbf{p})$. The justification of this argument can be seen in Network Reliability and Algebraic Structures [7].

This system of equations can easily be represented (see Appendix A) in matrix form. First define the matrix P similar to the adjacency matrix A, where the probability $p_e = p_{i,j}$ replaces it's respective 1. The system of equations is then equal t:

$$z = z \otimes P \oplus e_s$$

Eq. (5.4)

Where e_s is the s^{th} unit row vector with all component 0 except for the s^{th}, which is equal to 1[7].

5.2 Applying the Generalized Jacobi Method

The first iterative method examined in this section is the Generalized Jacobi Method. As mentioned in Shier's book [7], the "form of equations $z= z \otimes P \oplus e_s$ suggests an iterative solution scheme for calculating the reliability polynomials $R_{s,j}(\mathbf{p})$." If a starting estimate $z^{(0)} = e_s$ is used in the right hand side of equation 5.4 it will result in a new $z^{(1)}$. This step can then be repeated using the new z until some stopping criteria has been met. This iterative method is an extension of the Jacobi method with the use of the operations ⊕ and ⊗. Before continuing with this method the issue of convergence will be resolved.

Theorem: The successive iterates of the generalized Jacobi method converges to e_sA^* whenever $e_sA^* \geq z^{(0)}$. Where:

$$A^* = \oplus \sum_{j=0}^{n-1} A^j$$

e_sA^* is the minimal solution

Convergence is assured in at most n iterations.

The proof of this theorem can be seen in Network Reliability and Algebraic Structures [7]. Having proved that convergence is assured in at most n steps, we can now present the algorithm

- Algorithm 6: Generalized Jacobi method:

```
Construct the system of equations as shown above
Set $z^{(0)} = e_s$
For i = 1 to n
$z^{(i+1)}=z^{(i)} \otimes P \oplus e_s$
$R_{s,j}(\mathbf{p}) = z_j$
```

Use algorithm 5.4 to construct the proper probability vector **p** and numerical solve each $R_{s,j}(\mathbf{p})$.

One may note that after k iterations of the Generalized Jacobi Algorithm, the j-th element of the probability vector $z^{(k)}$ represents the probability of fire propagation from node s to node j subject to the restriction that only paths (joining node s to node j) of length k or less are considered. Paths of length (k + 1), if they exist, are considered in iteration (k + 1) to update the probability approximation. Consequently, vectors $z^{(k)}$ are non-decreasing and as k increases to n, the $z^{(k)}$ vectors approach the exact probability vector.

One possible advantage of this iterative scheme is that at the end of k iterations ($k < n$), $z^{(k)}$ represents a lower bound for the exact probability vector. If these lower bounds are good approximations, the algorithm may be halted after k iterations, to save expensive calculations. Naturally, it is important to determine a proper choice for the terminal value of k. To measure the closeness between the lower bound and the exact probability values, one needs to construct an upper bound as well (see section 6.0).

5.3 The List-directed Iterative Algorithm

The second iterative method examined in this section is called the List-directed iterative algorithm [7]. Unlike the generalized Jacobi method, this algorithm is specific to solving reliability polynomials. Recall from Section 1, the fundamental computation of the fire spread polynomial $R_{s,t}(\mathbf{p})$ is based on the $\oplus$ summing of all the paths that a fire could spread from node s to node t. Instead of finding all paths from the starting node to a node i and them $\oplus$ summing them, this method calculates $R_{s,j}(\mathbf{p})$ for each node i that can be reached with a path length at most k, where k is the current step number. Then in the next step the paths that go from the starting node to node i can be extended to node j using the edge (i,j).

Using the idea described in [7] an ordered list of nodes is kept, whose reliability polynomial has been recently updated. Once this list is empty, meaning no more changes are needed to be made to any polynomial $R_{s,i}(\mathbf{p})$, the method is terminated and each $R_{s,i}(\mathbf{p})$ is the reliability polynomial for each node i. Before discussing the mentioned "updating" we must first start with an initialization of the list.

```
Initialization:
for each node $j \neq$ the starting node
          if the edge k, connecting node s to node j exist
          then $L(j) = p_k$
          then add node j to the List
          else $L(j) = 0$
$L(s) = 1$
```

Going back to the example of Figure 2.4, with node 1 as the starting node, after the initialization we would have the following:

$List = \{2, 3\}$

$L(1) = 1$
$L(2) = p_1$
$L(3) = p_2$
$L(4) = 0$

Now that the initialization has been made we will discuss what is meant by the update of nodes. To do this we will describe the actual iterative steps that will be computed:

- Algorithm 7: The List-directed iterative method

```
while the List is not empty
          remove the first node i from the beginning (or end) of the List
          for each edge k that connects node i to another node j
                    Set $T = L(j) \oplus [L(i) \otimes p_k]$
                    If $T \neq L(j)$ then
                              $L(j) = T$
                              If j is not already in the List then add it
```

At the end of this algorithm the polynomials L(j) will be exactly equal to the desired reliability polynomials $R_{s,j}(\mathbf{p})$. To see how this algorithm works, we apply it to our example. Removing node i=2 from the list we will have two edges to deal with, edge 3 and 4.

$k = 3; j = 3$
$T = p_2 \oplus [p_1 \otimes p_3] = p_2 \oplus p_1 \otimes p_3$
$T \neq L(3)$ therefore $L(3) = p_2 \oplus p_1 \otimes p_3$; $j = 3$ is already in the list therefore it does not need to be added to the list.

$k = 4; j = 4$
$T = 0 \oplus [p_1 \otimes p_4] = p_1 \otimes p_4$
$T \neq L(4)$ therefore $L(4) = p_1 \otimes p_4$; $j = 4$ is not in the list and is therefore added

After removing node 2 we are left with the following:

List = {3, 4}

L(1) = 1
L(2) = p_1
L(3) = $p_2 \oplus p_1 \otimes p_3$
L(4) = $p_1 \otimes p_4$

If this method were continued, the List would be empty after one more iteration. We would then have the following value for L(4)

$$\begin{aligned} L(4) = R_{1,4}(\mathbf{p}) = {} & p_1p_4 + p_2p_5 + p_1p_3p_5 + p_2p_3p_4 - p_1p_2p_3p_4 - p_2p_3p_4p_5 \\ & - p_3p_4p_5p_1 - p_4p_5p_1p_2 - p_5p_1p_2p_3 + 2\, p_1p_2p_3p_4p_5 \end{aligned}$$

Eq. (5.5)

Which is exactly the reliability polynomial derive in Section 2.4 (Equation 2.13).

6. Upper and Lower Bounds

As the graphs of the ships get larger the number of paths involved and the reliability polynomials themselves get much larger too. Often the need to know the exact probability that a fire will spread from one compartment to another is not necessary. Perhaps an upper and lower bound on such probabilities will be sufficient if the error between them is small. In this section some methods for calculating both upper and lower bounds will be discussed.

6.1 Inclusion and Exclusion Bounds

The following is the method of inclusion-exclusion for calculating the probability of the union of a finite number of events:

$$P(\bigcup_{1}^{n} E_i) = \sum_{i=1}^{n} P(E_1) - \sum_{i<j}\sum^{n} P(E_1E_j) + \sum\sum_{i<j}\sum^{n} P(E_1E_jE_k) - \ldots + (-1)^{n-1} P(E_1E_2\ldots E_n)$$

Eq. (6.1)

A method of obtaining inclusion and exclusion bounds [4] using the above equation is the following:

$$P(\bigcup_{1=1}^{n} E_i) \le \sum_{i=1}^{n} P(E_1)$$

$$P(\bigcup_{1=1}^{n} E_i) \ge \sum_{i=1}^{n} P(E_1) - \sum_{i<j} \sum P(E_1 E_j)$$

$$P(\bigcup_{1=1}^{n} E_i) \le \sum_{i=1}^{n} P(E_1) - \sum_{i<j} \sum^{n} P(E_1 E_j) + \sum \sum_{i<j} \sum^{n} P(E_1 E_j E_k)$$

$$\ge \ldots$$

$$\le \ldots$$

Eq. (6.2)

where the inequality always changes direction as each addition term of the expansion in Equation 6.1 is added.

In order to apply this method of bounds to the application of fire safety we must define our events E_i. We can think of the event E_i as the event that an individual path will occur. Therefore a fire will spread from the starting node to the target node if and only if it spreads across at least one of the paths, or at least one of the events E_i occur:

$$R_{s,t}(\mathbf{p}) = P(\bigcup_{1}^{n} E_i)$$

Eq. (6.3)

where n is the total number of paths from the starting node to the target node. By defining the following:

$$P(E_i) = \prod_{l \in path\, i} p_l$$

$$P(E_i E_j) = \prod_{l \in path\ i\ or\ path\ j} p_l$$

$$P(E_i E_j E_k) = \prod_{l \in path\ i\ or\ path\ j\ or\ path\ k} p_l$$

Eq. (6.4)

we can apply the Equations 6.2 to obtain upper and lowers bounds of the probability that a fire will spread from the starting node to the target node. By adding and subtracting more terms we can obtain both upper and lower bounds whose difference is within a certain threshold.

It must be noted that as the number of terms (on the right hand side of Equations 6.1), considered in deriving these bounds increase, so does the complexity in computations. It remains to be seen, how useful these bounds would prove to be.

7. Implementation of Time and L-Bar Dependencies

Two areas that require further implementation are time dependency and L-bar dependency. These ideas are closely related. These both have an impact on the calculation of the probability vector **p**. In this section, the implementation of these ideas will be discussed.

7.1 L-bar Dependency

As mentioned in Section 4.2, the probabilities that a fire will spread from one node to two other adjacent nodes are dependent. This dependency arises from the fact that in order for the fire to spread out of the current compartment it has to reach the L-bar state. Once it has reached the L-bar state the fire then has a possibility of spreading to any adjacent room. Recall the following figure:

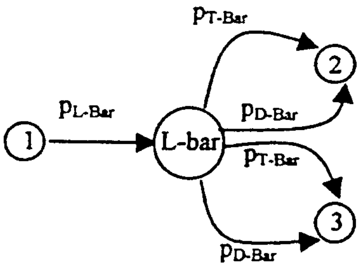


Figure 7.1: Detailed Representation of Three Nodes, Two Edges

Here in order to spread from node 1 to either node 2 to node 3 the fire must "spread" to the L-bar state. In order to implement this idea of L-bar dependency the L-bar nodes need to be added to the input graph. To do this, the original adjacency matrix must be transformed.

7.1.1 Transforming the Adjacency Matrix

When a user inputs the adjacency matrix, it is an n x n matrix where n is the number of nodes in the graph. Notice that each node representing a different compartment must have it's own L-bar state. Therefore n more nodes will be added to the graph, therefore making the new adjacency matrix a 2n x 2n matrix.

The next step will be to map the original nodes to the new adjacency matrix. In the new matrix each original node will be represented by the following formula.

new node index = (old node index * 2) – 1

Eq. (7.1)

This will leave the even indices in the new adjacency matrix representing the corresponding L-bar nodes. The kth index in the new adjacency matrix, if even, will represent the L-bar state for the corresponding k/2 index in the original matrix.

The last step is to translate the positive entries in the old adjacency matrix to the new adjacency matrix. Again, if a fire is going to spread from the ith node to an adjacent jth node it must first "spread" to the ith L-bar state, and then to the jth node. Therefore for every positive entry representing an edge in the original adjacency matrix, two positive entries representing two edges will be added to the new matrix. This can be done using the following formula. If A(i,j) = 1 in the original matrix:

$$A_{New}(2i-1, 2i) = 1$$
$$A_{New}(2i, 2j-1) = 1$$

Eq. (7.2)

Notice that in the new adjacency matrix, $A_{New}(2i-1, 2i)$ represents the edge that connects the ith node in the original matrix to the ith L-bar state. Similarly, $A_{New}(2i, 2j-1)$ represents the edge connecting the ith L-bar state to the jth node in the original matrix.

For example, for the following adjacency matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Eq. (7.3)

The new adjacency matrix would be:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Once this transformation has been made the rest of the computer implementation is unchanged except for the calculation of **p**.

If given p_k, its numerical value can be found in the following manner. First translate the index k to it's corresponding ordered pair (i,j). Now:

If j is odd

$p_k = p_{L\text{-bar}}$ for the j/2 node in the original graph.

If j is even

$p_k = p_{T\text{-bar}} \oplus p_{D\text{-bar}} = p_{T\text{-bar}} + p_{D\text{-bar}} - p_{T\text{-bar}}p_{D\text{-bar}}$
for the barrier separating the i/2 and (j+1)/2 node in the original graph.

7.2 Time Dependency

As a fire spreads through a ship, the probability of it spreading from compartment to compartment depends on time. This dependency arises from the fact that the $p_{L\text{-bar}}$, $p_{T\text{-bar}}$, and $p_{D\text{-bar}}$ are changing with time due to the stage of fire growth and changing I, A, and M values.

Since the construction of the reliability function between any two nodes will not depend on time, the first three algorithms (*Input, Finding all paths of length k or less, and Calculating the ⊕sum of a list of paths*) do not need to be changed. In order to implement time dependency a change was made to the fourth Algorithm, *Calculating the numerical value of $R_{s,t}(\mathbf{p})$*.

The fourth Algorithm, *Calculating the numerical value of $R_{s,t}(\mathbf{p})$*, first calculates the probability vector **p**. In order to calculate this vector time needs to be consider. Recall from the previous section

If j is even

$p_k = p_{T\text{-bar}} \oplus p_{D\text{-bar}} = p_{T\text{-bar}} + p_{D\text{-bar}} - p_{T\text{-bar}}p_{D\text{-bar}}$
for the barrier separating the i/2 and (j+1)/2 node in the original graph.

But the value

$$p_k = p_{T\text{-bar}} \oplus p_{D\text{-bar}} = p_{T\text{-bar}} + p_{D\text{-bar}} - p_{T\text{-bar}}p_{D\text{-bar}}$$

is dependent on time. So to incorporate time dependency we simply need to input a time and then calculate the above value using the inputted time.

8. Conclusion

In this report it was shown how the mathematical tools of reliability theory can be applied to fire safety analysis. These tools come from the specific area of reliability for interconnected systems, a highly studied area of mathematics.

The keys to this computation are the two operations ⊕ and ⊗. These operations model combinations of paths from a source node to a target node, taking into careful consideration how a fire might spread along these paths. These operations construct the reliability function for starting and target nodes. The function constructed depends on the probability vector, **p**. This time dependent probability vector is obtained by engineering methods in fire safety analysis.

Algorithms for the computer implementation of this method are developed. These algorithms include a way to construct the complete set of paths between the starting and the target nodes, with a specified bound on the paths' length. Also included are algorithms for the implementation of the $\oplus$ and $\otimes$ operations which determine the value of the reliability polynomial for a given probability vector.

References

[1] Clouthier, Elizabeth; Rich, Doris; and Romberg, Betty, Ship Applied Fire Engineering (SAFE) User Manual Version 2.2, A Computer Model for the Implementation of the Ship Fire Safety Engineering Methodology (SFSEM), Report No. CG-D-10-96, U.S. Coast Guard, Final Report, March 1996.

[2] Fitzgerald, Robert W., Building Firesafety Engineering Method: A Workbook, Preliminary Draft, March 1993

[3] Sprague, Chester M. and Dolph, Brian, Theoretical Basis of the Ship Fire Safety Engineering Methodology, Report No. CG-D-30-96, U.S. Coast Guard, Final Report, September 1996.

[4] Ross, Sheldon, Introduction to Probability Models, Academic Press, 1993

[5] Brown, Kevin, http://www.seanet.com/~ksbrown/kmath303.htm, accessed May 5, 1997

[6] Bondy, J. A. and Murty, U. S. R., Graph Theory with Applications, American Elsevier Publishing Company, 1976

[7] Shier, Douglas, Network Reliability and Algebraic Structures, Clarendon Press, 1991

[8] Ross, Sheldon, A First Course in Probability, Macmillan College Publishing Company, 1994

[9] Sprague, Chester M.; White, Derek; and Dolph, Brian, Fire Safety Analysis of the 87' Coastal Patrol Boat (CPB), Technical Report, Pending Approval for Publication in the NTIS, June 1997.

[10] Sprague, Chester M., The Cumulative L-Curve Algorithm in the Ship Fire Safety Engineering Methodology, U.S. Coast Guard, Internal Report, April, 1996.

Appendix A: Properties of the operations ⊕ and ⊗

Lemma 1.

Let f, g, h be arbitrary reliability functions, then:

(i) $f \oplus f = f$

(ii) $f \oplus g = g \oplus f$

(iii) $f \oplus (g \oplus h) = (f \oplus g) \oplus h$

(iv) $f \otimes f = f$

(v) $f \otimes g = g \otimes f$

(vi) $f \otimes (g \otimes h) = (f \otimes g) \otimes h$

(vii) $f \oplus (f \otimes g) = f$

(viii) $f \otimes (f \oplus g) = f$

(ix) $f \oplus (g \otimes h) = (f \oplus g) \otimes (f \oplus h)$

(x) $f \otimes (g \oplus h) = (f \otimes g) \oplus (f \otimes h)$

(xi) $f \oplus 0 = f$ where 0 is the probability of a path that will never occur

(xii) $f \otimes 0 = 0$ where 0 is the probability of a path that will never occur

(xiii) $f \oplus 1 = 1$

(xiv) $f \otimes 1 = f$ where 1 is the probability of a path that will always occur

Proof:

(i) $f \oplus f = f + f - f \otimes f = f + f - f = f$

(ii) $f \oplus g = f + g - f \otimes g = g + f - g \otimes f = \mathrm{g} \oplus f$

(iii) $f \oplus (g \oplus h) = f + (g + h - g \otimes h) - f \otimes (g + h - g \otimes h) = (f + g - f \otimes g) +$ h - $(f + g - f \otimes g) \otimes \mathrm{h}) = (f \oplus g) \oplus h$

(iv) By definition

(v) By definition

(vi) By definition

(vii) $f \oplus (f \otimes g) = f + (f \otimes g) - f \otimes (f \otimes g) = f + (f \otimes g) - (f \otimes g) = f$

(viii) $f \otimes (f \oplus g) = f \otimes (f + g - f \otimes \mathrm{g}) = f + f \otimes g - f \otimes g = f$

(ix) $f \oplus (g \otimes h) = f + (g \otimes h) - f \otimes (g \otimes h) = (f + g - f \otimes g) \otimes (f + h - f \otimes h)$ $= (f \oplus g) \otimes (f \oplus h)$

(x) $f \otimes (g \oplus h) = f \otimes (g + h - g \otimes h) = f \otimes g + f \otimes h - f \otimes g \otimes h = (f \otimes g) \oplus$ $(f \otimes h)$

(xi) $f \oplus 0 = f + 0 - f \otimes 0 = f$

(xii) The probability of any event A intersected with the event that nothing will happen is always equal to 0.

(xiii) $f \oplus 1 = f + 1 - f \otimes 1 = f + 1 - f = 1$

(xiv) The probability of any event A intersected with the event of probability 1 equal to the probability of the event A.

[BLANK]

Appendix B: User's Guide

This Appendix includes a brief description on how to use the included program.

Step 1: Creating the lbar.txt file

Each of the compartments in the ship have a probability of reaching the L-bar state. These probabilities need to be put into a file in the following format.

```
4
0.1
0.5
0.34
0.4
```

The first number represents the number of L-bar probabilities or compartments in the ship. The next numbers are the actual probabilities. In other words, the probability that the 1st compartment will reach the L-bar state is 0.1, the probability that the 2nd compartment will reach the L-bar state is 0.5, and so on.

Step 2: Creating the materials.txt file

Inside a cutter there are several types of materials that separate different compartments. Each of these materials have different data sets used to calculate their respective T-bar and D-bar probabilities. T-bar and D-bar data for barrier materials is provided in Appendix A of the SAFE User Manual. This data needs to be stored into a file in order for the program to use this data. The file used in the example discussed in this report is (the material ID and Description has been added for clarity):

Material.txt file							ID	Material Description
16								
1	0	0	0	0	0	0	000	Zero Strength
2	3	6	10	3	6	10	A2I	¼" Aluminum with thermal insulation
3	0	2	4	4	6	10	A2U	¼" Aluminum
4	1	3	4	1	3	4	C5U	5/8" Celotex
5	2	5	7	25	35	40	F2U	¼" Fiberglas
6	2	8	10	9	18	22	NPI	Nomex honeycomb - (plastic laminate & Insulation)
7	2	6	14	3	12	20	NPU	Nomex honeycomb - (plastic laminate facing)
8	8	20	30	55	80	105	NSU	Nomex honeycomb - (stainless steel facing)
9	6	12	21	10	20	27	P7P	7/8" Plywood - (plastic laminate facing)
10	5	15	18	75	100	120	S2I	¼" Steel with thermal insulation
11	1	4	10	60	80	100	S2U	¼" Steel
12	6	18	20	80	110	130	S3I	3/8" Steel with thermal insulation
13	1	4	10	65	85	105	S3U	3/8" Steel
14	6	18	20	80	110	130	S4I	½" Steel with thermal insulation
15	2	5	12	70	90	110	S4U	½" Steel
16	2	5	12	75	95	115	S5U	5/8" Steel

Similar to some of the other files, the first number represents the number of materials. Then each line represents a different material. The first number in each line is the index number of the material. The next 3 numbers is the T-bar data and the last three are the D-bar data.

Step 3: Creating the adj.txt file

This file will contain the adjacency matrix, which represents the graph whose reliability polynomial will be found. The first number in the file should be the dimension of the matrix. The next line should contain the first row of the matrix, the third line should contain the second row of the matrix, and so on until all rows of the matrix are in the file. The adj.txt file for Figure 2.4 is

```
4
0 1 2 0
1 0 3 4
2 3 0 4
0 4 4 0
```

In order to represent which material(s) separates two rooms the index number of the material in the material.txt file is used. For example, room 1 and 3 are separated by material number 2 because there is a 2 in the (1, 3) position in the matrix.

Step 4: Executing the program

At the command prompt run the executable file. For example:

```
C:\>analysis
```

Step 4: Answering necessary input

During the execution of the program several questions will need to be answered. The first will be a warning

```
WARNING! WARNING! WARNING! WARNING! WARNING!

    The following files will be deleted:
    paths.txt
    solution.txt
    temp.txt

    Would you like to continue (1-Yes 2-No) ?
    (If results from a previous analysis should be saved, do so at this time.)
```

The second will be to

> "Enter the filename containing the adjacency matrix:"

The third will be to

> "Enter the starting node: "

The fourth will be to

> "Enter the target node: "

The fifth will be to

> "Enter maximum path length: "

The last will be to

> "Enter the # of timesteps: "

After these step have been following the program will output the probability at each time step that a fire will spread from the starting node to the target node.

[BLANK]

Appendix C: Program Code

```
/*
    Title: analyis.c
    Created by: Adam T. Garland
    Last Revised: June 19, 1997

    Description:
    This program reads in an adjacency and probability matrix
    of a graph.  It then finds all the paths of length k or less
    between the desired two node.  The reliability function is then
    computed and then the numerical value for the probability that
    a fire spread between the desired node is computed.
*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define MAXPLENGTH   100
#define MAXSLENGTH   100
#define MAXARCS      100
#define MAXNODES 100
#define MAX          1000
#define MAXQ      10000
/*#define MAXPATHL    4*/

int A[MAXARCS][MAXARCS];
int Aold[MAXARCS][MAXARCS];
int Edges[MAXARCS][MAXARCS];
int OldEdges[MAXARCS][MAXARCS];
int IndexI[MAXARCS];
int IndexJ[MAXARCS];
int N;
int tedges;

void main (void)
{

    void input (void);
    void findpaths (int, int);
    void solution (void);
    double value (int);
    double answer;

    int cont,
        start,
        target,
        time,
        timesteps;



    printf("\nWelcome, starting ship analysis");
    printf("\n\nWARNING! WARNING! WARNING! WARNING! WARNING!
WARNING!");
    printf("\n\nThe following files will be deleted: ");
    printf("\npaths.txt");
    printf("\nsolution.txt");
    printf("\ntemp.txt");
    printf("\n\nWould you like to continue (1-Yes 2-No) ? ");
    scanf("%d", &cont);

    if(cont==1)
    {

        system("del temp.txt");
        system("del solution.txt");
        system("del paths.txt");

        printf("\nGathering Input");
        input();

        printf("\nEnter the starting node: ");
        scanf("%d", &start);
        printf("\nEnter the target node: ");
        scanf("%d", &target);

        start = (start*2)-1;
        target = (target*2)-1;

        printf("\nFinding Paths");
        findpaths(start, target);

        printf("\n(+) summing paths");
        solution();

        printf("\nEnter the # of timesteps: ");
        scanf("%d", &timesteps);
        printf("\nFinding numerical value(s)");
        printf("\nTime      Probability\n");
        for(time=1; time<=timesteps; time++)
        {
            answer = value(time);
            printf("\n%d  %lf", time, answer);
        }
    }
    else
    {
        printf("\n\nYou have choosen not to continue\n");
    }

    printf("\n");
}

/*
    This subroutine finds all paths of length k or less
    in the graph inputed.
*/

/*
    This subroutine reads in the adjacency matrix
    from the filename inputed adj.txt
*/

void input(void)
{

    FILE *fpA;
    int i,
        j,
        count;

    char filename[20];

    printf("\nEnter the filename containing the adjacency matrix: ");
    scanf("%s", filename);

    fpA = fopen(filename, "r");

    count = 1;

    fscanf(fpA, "%d", &N);
```

```
printf("\nYou have entered the follow %d x %d matrix\n", N, N);

/*Read is the adj matrix from the file adj.txt*/

for (i=1; i<=N; i++)
{
     for (j=1; j<=N; j++)
     {
          fscanf(fpA, "%d", &Aold[i][j]);
          printf("%d ", Aold[i][j]);
          if(Aold[i][j]!=0)
          {
               OldEdges[i][j]=count;
               count = count+1;
          }
     }
     printf("\n");
}

for (i=1; i<=N; i++)
{
     for (j=1; j<=N; j++)
     {
          if(Aold[i][j]!=0)
          {
               A[(2*i)-1][2*i]=Aold[i][j];
               A[2*i][(2*j)-1]=Aold[i][j];
          }
     }
}

N=2*N;

count = 1;
printf("\nThe new Adjacency Matrix is:\n");
for (i=1; i<=N; i++)
{
     for (j=1; j<=N; j++)
     {
          printf("%d", A[i][j]);
          /*Numbers the edges*/

          if (A[i][j]!=0)
          {
```

```
                    Edges[i][j] = count;
                    IndexI[count]=i;
                    IndexJ[count]=j;
                    count = count + 1;
                }
            }
            printf("\n");
        }
        /*Keeps track of the total number of Edges */

        tedges = count-1;
        printf("\nThe total number of edges = %d", tedges);

        fclose(fpA);
}

/*
        This subroutine finds all paths of length k or less
        in the graph inputed.
*/

void findpaths (int start, int target)
{

        FILE *fpPath;

        int i,
            j,
            k,
            add,
            cont,
            parent,
            position,
            total,
            count,
            MAXPATHL,
            path[MAXARCS],
            arcpath[MAXARCS],
            temp,
            Neigh[MAXNODES][MAXARCS],
            Q[MAXQ][3];


        fpPath = fopen("paths.txt", "w");
```

```
printf("\nEnter maximum path length: ");
scanf("%d", &MAXPATHL);

MAXPATHL = MAXPATHL *2;

/*Finds all the neighbors of each node*/

count = 1;
for (i=1; i<=N; i++)
{
     for (j=1; j<=N; j++)
     {

          if (A[i][j]!=0 && j!=start)
          {
               Neigh[i][count]= j;
               count = count + 1;
          }
     }

     /*Marks end of the neighbors*/

     Neigh[i][count]= -1;
     count = 1;
}




/*Q[Position in Queue][0] = Node
  Q[Position in Queue][1] = Parent Position
  Q[Position in Queue][2] = Level*/

Q[1][0] = start;
Q[1][1] = -1;
Q[1][2] = 1;

parent =1;
position = 2;

/*The following while loop creates the queue*/
```

```
cont = 1;

while (parent < position)
{
     i = Q[parent][0];
     if(i!=target && Q[parent][2] < MAXPATHL)
     {
     j=1;
          while (Neigh[i][j] != -1)
          {

               add=1;
               k = parent;

               /* Check is node to be added is already in branch */

               while(Q[k][1]!=-1)
               {
                    if (Q[k][0] == Neigh[i][j])
                    {
                         add = 0;
                    }

                    k=Q[k][1];
               }

               if(add==1)
               {
                    Q[position][0] = Neigh[i][j];
                    Q[position][1] = parent;
                    Q[position][2] = Q[parent][2]+1;

                    position = position + 1;

               }


               j = j + 1;

          }
     }

     parent = parent + 1;
}
```

```
	/*Marks the end of the queue*/
	Q[position][0] = -1;
	Q[position][1] = -1;

	i=1;
	total=0;

	while(Q[i][0]!=-1)
	{
		if(Q[i][0]==target)
		{
			total = total + 1;
		}
		i=i+1;
	}

	printf("\n\nThe total number of paths = %d and are: ", total);

	fprintf(fpPath, "%d", total);

	i=1;

/*
	This while loop extracts the paths from the queue created
*/

	while(Q[i][0]!=-1)
	{
		if(Q[i][0]==target)
		{
			k = i;

			for(j=Q[i][2]; j>0; j--)
			{
				path[j]=Q[k][0];
				k=Q[k][1];
			}

			printf("\nNode path ");

			for(j=1; j<=Q[i][2]; j++)
			{
				printf("%d ", path[j]);
			}
```

```
            printf("with arc path: ");

            fprintf(fpPath, "\n%d 1 ", (Q[i][2]-1));

            /*This finds the arc path from the know node path*/

            for(j=1; j<Q[i][2]; j++)
            {
                arcpath[j]= Edges[path[j]][path[j+1]];
            }

            /*This sorts the arcpath in ascending order*/


            for(k=1; k<Q[i][2]; k++)
            {
                for(j=k+1; j<Q[i][2]; j++)
                {
                    if(arcpath[k]>arcpath[j])
                    {
                        temp = arcpath[k];
                        arcpath[k] = arcpath[j];
                        arcpath[j] = temp;
                    }
                }
            }

            /*This prints the arc path to the screen and file*/

            for(j=1; j<Q[i][2]; j++)
            {
                printf("%d ", arcpath[j]);
                fprintf(fpPath, "%d ", arcpath[j]);
            }


        }
        i=i+1;
    }

    printf("\n");
    fclose(fpPath);

}
```

```
/*
    This subroutine (+) sums all the paths in the path.txt file
*/

void solution (void)
{

    FILE *fpPath, *fpSol, *fpTemp;

    int *circlex (int, int, int [MAXSLENGTH], int [MAXSLENGTH]);

    int i,
        j,
        k,
        nPath,
        nSol,
        nTemp,
        sPath,
        sSol,
        sTemp,
        N,
        tempPath[MAXPLENGTH],
        tempSol[MAXSLENGTH],
        temp[MAXSLENGTH],
        *pTemp;


    fpPath = fopen("paths.txt", "r");
    fpSol = fopen("solution.txt", "a+");
    fpTemp = fopen("temp.txt", "w+");


    fscanf(fpPath, "%d", &N);

    /*Reads in the length of current path*/
    fscanf(fpPath, "%d", &nPath);
    fscanf(fpPath, "%d", &sPath);

    /*Prints the length of current path to file*/
    fprintf(fpSol, "%d ", nPath);
    fprintf(fpSol, "%d ", sPath);
```

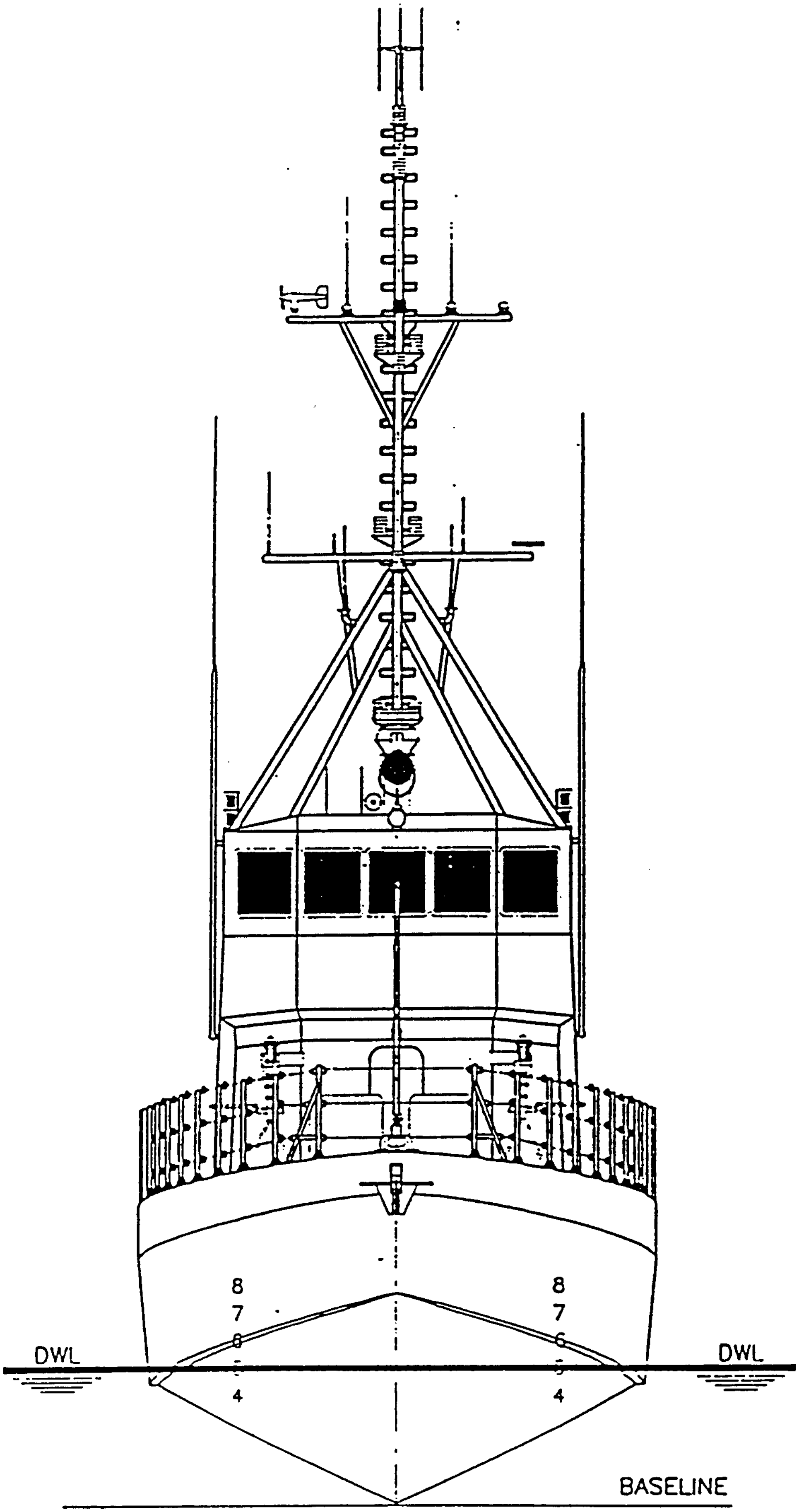
```
for (k=1; k<=nPath; k++)
{
    fscanf(fpPath, "%d", &tempPath[k]);
    fprintf(fpSol, "%d ", tempPath[k]);
}

fprintf(fpSol, "\n");

for(i=2; i<=N; i++)
{


    /*Reads in the length of current path*/
    fscanf(fpPath, "%d", &nPath);
    fscanf(fpPath, "%d", &sPath);

    /*Prints the length of current path to file*/
    fprintf(fpTemp, "%d ", nPath);
    fprintf(fpTemp, "%d ", sPath);


    /*This loop reads in current path and prints in to the file*/
    for (k=1; k<=nPath; k++)
    {
        fscanf(fpPath, "%d", &tempPath[k]);
        /*printf("%d ", tempPath[k]);*/
        fprintf(fpTemp, "%d ", tempPath[k]);
    }

    fprintf(fpTemp, "\n");

    rewind(fpSol);

    for(j=1; j<=(pow(2, (i-1))-1); j++)
    {
        /*Reads in the length of current path*/
        fscanf(fpSol, "%d", &nSol);
        fscanf(fpSol, "%d", &sSol);

        /*This loop reads in current path and prints in to the file*/
        for (k=1; k<=nSol; k++)
        {
            fscanf(fpSol, "%d", &tempSol[k]);
        }
    pTemp = circlex(nPath, nSol, tempPath, tempSol);

    k=1;

    while(pTemp[k] != 0)
    {
        k=k+1;
    }

    k=k-1;

    fprintf(fpTemp,"%d ", k);

    if(sPath==1 && sSol==1)
    {
        fprintf(fpTemp,"-1 ");
    }
    else if(sPath==(-1) && sSol==1)
    {
        fprintf(fpTemp,"1 ", k);
    }
    else if(sPath==1 && sSol==(-1))
    {
        fprintf(fpTemp,"1 ", k);
    }
    else if(sPath==(-1) && sSol==(-1))
    {
        fprintf(fpTemp,"-1 ", k);
    }


    k=1;

    while(pTemp[k] != 0)
    {
        fprintf(fpTemp, "%d ", pTemp[k]);
        k=k+1;
    }

    fprintf(fpTemp, "\n");
}


/*
```

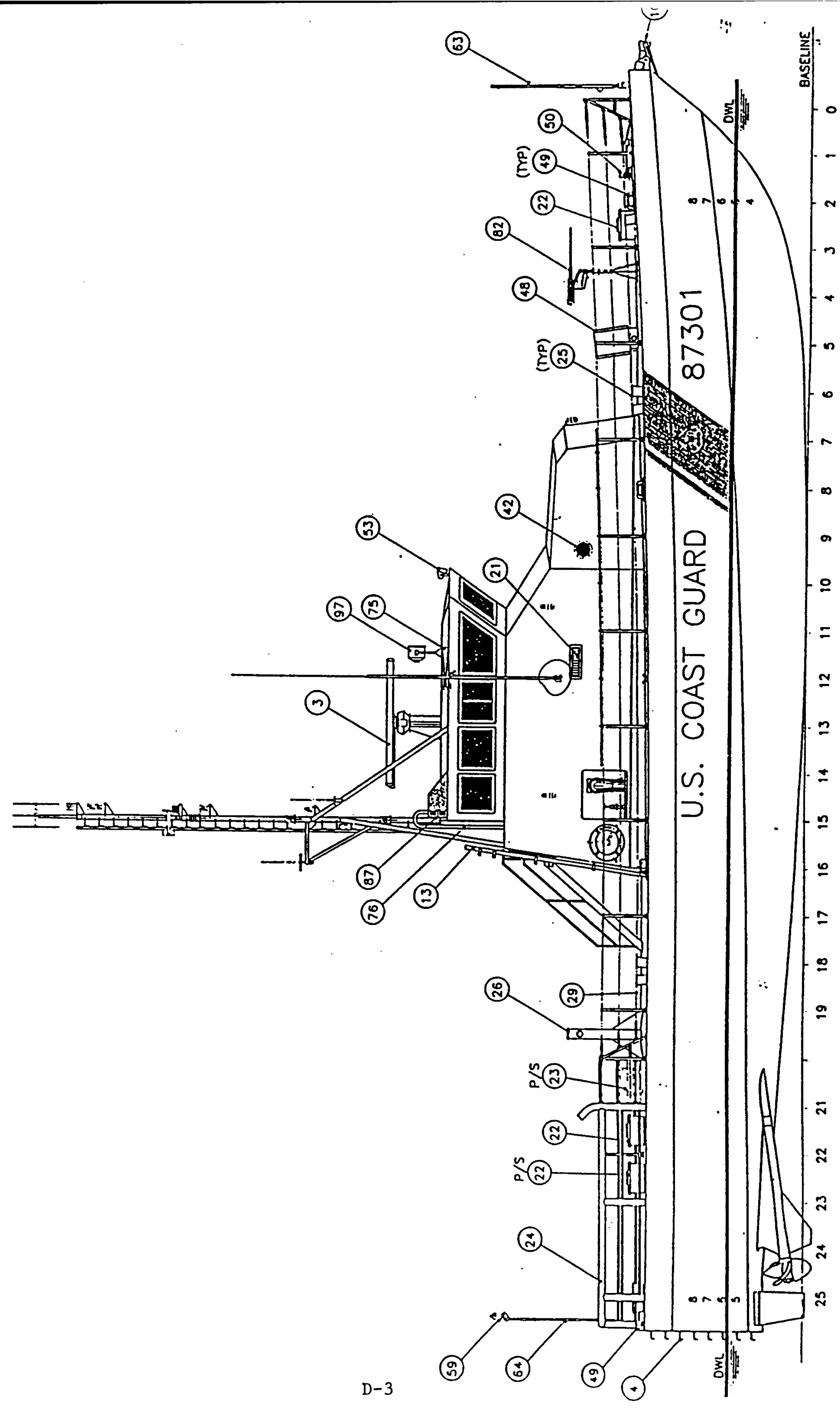
```
         This moves the info from temp to solution
        */

        rewind(fpTemp);
        fseek(fpSol, 1, SEEK_END);

        j=0;

        for(j=1; j<=(pow(2, (i-1))); j++)
        {
            /*Reads in the length of current temp*/
            fscanf(fpTemp, "%d", &nTemp);
            fscanf(fpTemp, "%d", &sTemp);
            fprintf(fpSol, "%d ", nTemp);
            fprintf(fpSol, "%d ", sTemp);

            /*Reads in the current sol*/
            for (k=1; k<=nTemp; k++)
            {
                fscanf(fpTemp, "%d", &temp[k]);
                fprintf(fpSol, "%d ", temp[k]);
            }

            fprintf(fpSol,"\n");
        }

        rewind(fpTemp);
    }



    fprintf(fpSol,"99999\n");

    fclose(fpPath);
    fclose(fpSol);
    fclose(fpTemp);


}

/*
    This subrountine performs the (x) operation
*/
```

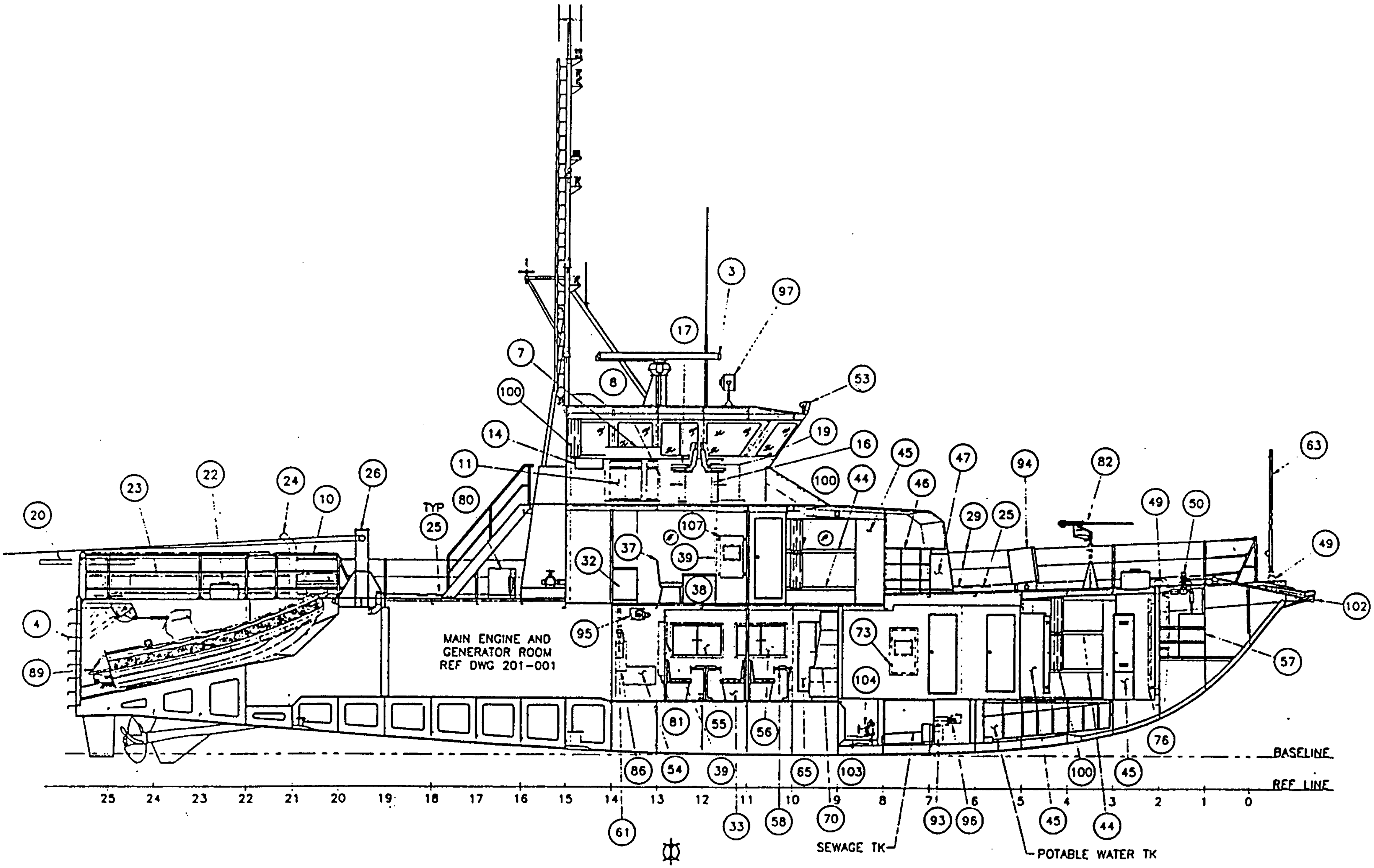
```
int *circlex (int m, int n, int listm[], int listn[])
{

    int i,
        j,
        count,
        *temp;

        temp = malloc(sizeof(int)*MAXSLENGTH);

    i=1;
    j=1;
    count=1;

        while(i<=m && j<=n)
        {
            if(listm[i]==listn[j])
            {
                temp[count] = listm[i];
                i=i+1;
                j=j+1;
                count = count+1;
            }
            else if (listm[i]>listn[j])
            {
                temp[count] = listn[j];
                j=j+1;
                count=count+1;
            }
            else
            {
                temp[count] = listm[i];
                i=i+1;
                count=count+1;
            }
        }

        if(i>m)
        {
            for(i=j; i<=n; i++)
            {
                temp[count]=listn[i];
                count=count+1;
            }
        }
        else
        {
            for(j=i; j<=m; j++)
            {
                temp[count]=listm[j];
                count=count+1;
            }
        }

        temp[count]=0;

    count=count-1;


    return (temp);
}


/*
    This subroutine computes the numerical value for the reliability polynomial
*/

double value (int time)
{

    FILE *fpSol, *fpProb, *fpMat, *fpLbar;

    double fail(int, int, int, int);

    int i,
        j,
        k,
        l,
        count,
        material,
        nPath,
        nMat,
        nLbar,
        round,
        sign,
        x[MAXARCS][7];

    double  probs[MAXARCS],
            Lbar[MAXARCS],
            tbar,
        dbar,
        pathprob,
        answer;

fpSol = fopen("solution.txt", "r");
fpProb = fopen("probs.txt", "r");
fpMat = fopen("materials.txt", "r");
fpLbar = fopen("lbar.txt", "r");

answer = 0;
count = 1;

fscanf(fpMat, "%d", &nMat);
fscanf(fpLbar, "%d", &nLbar);

for (i=1; i<=nMat; i++)
{
    fscanf(fpMat, "%d", &j);
    fscanf(fpMat, "%d", &x[j][1]);
    fscanf(fpMat, "%d", &x[j][2]);
    fscanf(fpMat, "%d", &x[j][3]);
    fscanf(fpMat, "%d", &x[j][4]);
    fscanf(fpMat, "%d", &x[j][5]);
    fscanf(fpMat, "%d", &x[j][6]);
}


for(i=1; i<=nLbar; i++)
{
    fscanf(fpLbar, "%lf", &Lbar[i]);
}

count=1;
for(i=1; i<=N; i++)
{
    for(j=1; j<=N; j++)
    {

        if (Aold[i][j]!=0)
        {
            material=Aold[i][j];
            tbar = fail(time, x[material][1], x[material][2], x[material][3]);
            dbar = fail(time, x[material][4], x[material][5], x[material][6]);
            probs[count]= tbar + dbar - (tbar*dbar);
            count = count + 1;
        }

    }

}


fscanf(fpSol, "%d", &nPath);

while(nPath != 99999)
{

    fscanf(fpSol, "%d", &sign);

    pathprob=1.0;
    l=1;

    while(l<=nPath)
    {
        fscanf(fpSol, "%d", &k);

        round = IndexJ[k]/2;

        if(IndexJ[k] == (round*2)) /*even*/
        {
            pathprob = pathprob*Lbar[round];
            l=l+1;
        }

        if(IndexJ[k] != (round*2)) /*odd*/
        {
            i = IndexI[k];
            j = IndexJ[k];
            count = OldEdges[i/2][(j+1)/2];
            pathprob = pathprob*probs[count];
            l=l+1;
        }
    }

    if(sign==1)
    {
        answer = answer + pathprob;
    }
    else
    {
            answer = answer - pathprob;
        }

        fscanf(fpSol, "%d", &nPath);
    }

    return(answer);

}


double fail(int t, int a, int b, int c)
{

    double answer;


    if (t >= c)
    {
        answer = 100.0;
    }
    else if (t >= b)
    {
        answer = ((50/(c-b))*(t-b)+50);
    }
    else if (t >= a)
    {
        answer = ((50/(b-a))*(t-a));
    }
    else
    {
        answer = 0.0;
    }

    answer = answer/100;

    return(answer);

}
```

Appendix D: Layout of a U.S. Coastal Patrol Boat

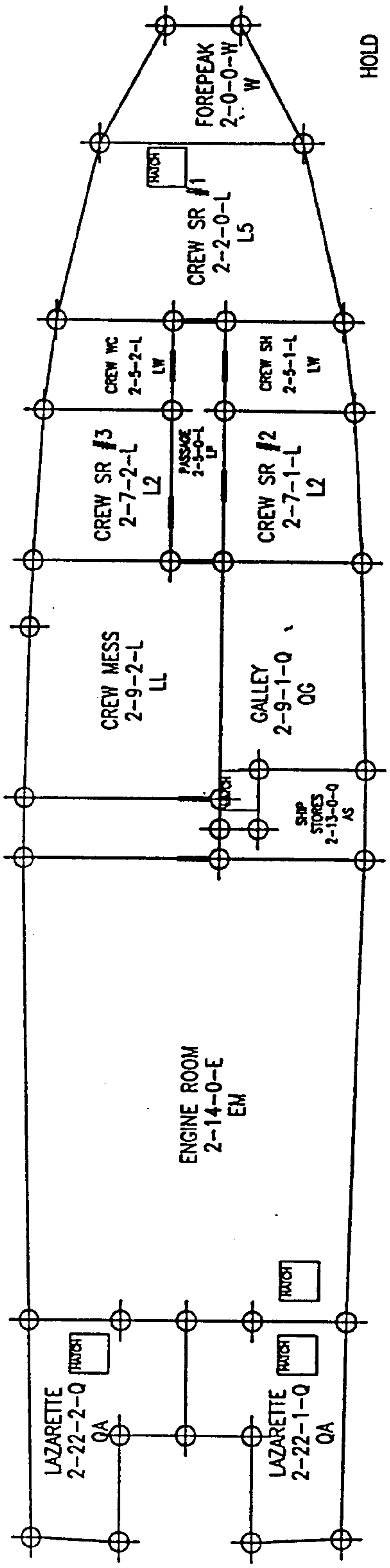
This appendix includes the profile and plan views of all decks in the 87' Coastal Patrol Boat. The plan views include the access fittings for each compartment such as doors, scuttles, hatches, and operable windows. The compartmentation shown represents how the ship was modeled in AutoCAD for the fire safety analysis.

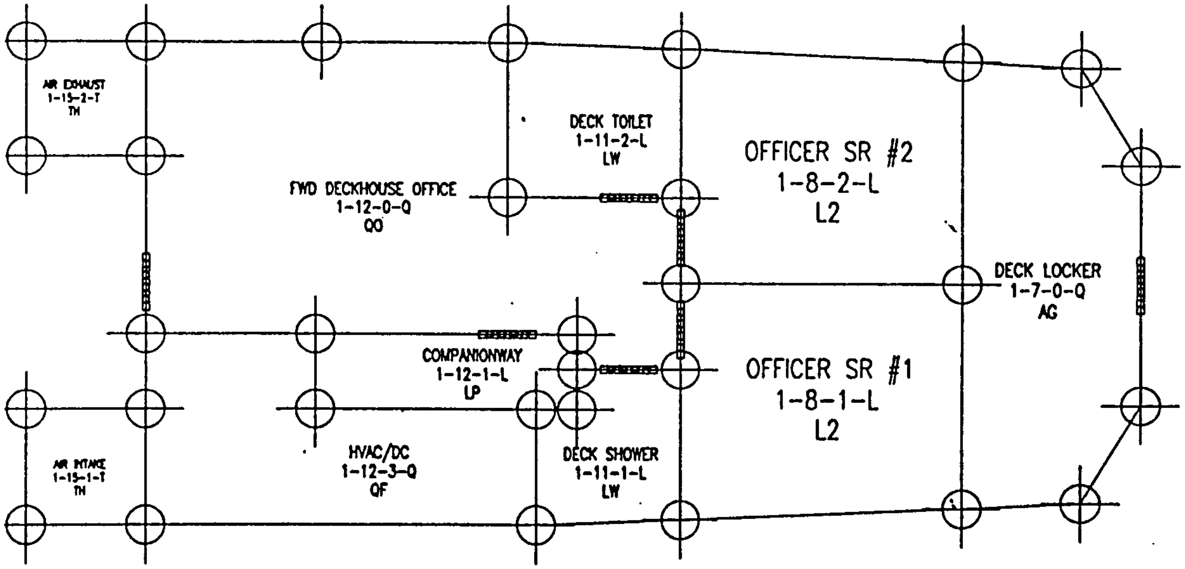

8
7
6
5
4
8
7
6
5
4
DWL
DWL
BASELINE


63
50
(TYP) 49
22
82
48
(TYP) 25
42
53
21
97
75
3
87
76
13
26
29
P/S 23
22
P/S 22
24
59
64
49
4
DWL
DWL
BASELINE
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
21
22
23
24
25
8
7
6
5
4
U.S. COAST GUARD
87301

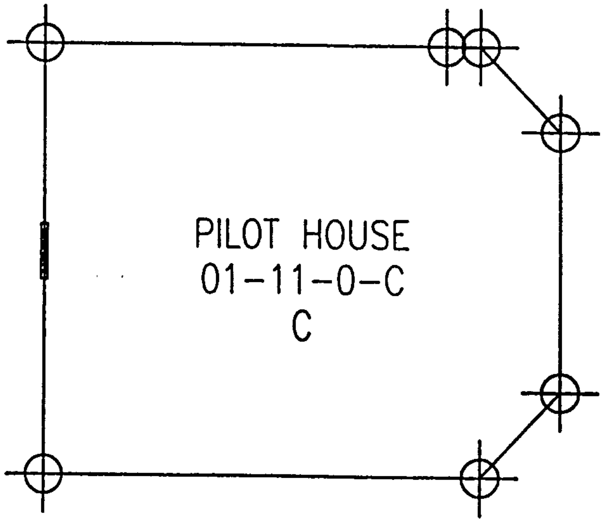


INBOARD PROFILE


HOLD
FOREPEAK
2-0-0-W
W
HATCH
CREW SR #1
2-2-0-L
L5
CREW WC
2-5-2-L
LW
CREW SH
2-5-1-L
LW
CREW SR #3
2-7-2-L
L2
PASSAGE
2-5-0-L
LP
CREW SR #2
2-7-1-L
L2
CREW MESS
2-9-2-L
LL
GALLEY
2-9-1-Q
QG
HATCH
SHIP
STORES
2-13-0-Q
AS
ENGINE ROOM
2-14-0-E
EM
HATCH
HATCH
HATCH
LAZARETTE
2-22-2-Q
QA
LAZARETTE
2-22-1-Q
QA


AIR EXHAUST
1-15-2-T
TH
FWD DECKHOUSE OFFICE
1-12-0-Q
QO
DECK TOILET
1-11-2-L
LW
OFFICER SR #2
1-8-2-L
L2
DECK LOCKER
1-7-0-Q
AG
COMPANIONWAY
1-12-1-L
LP
OFFICER SR #1
1-8-1-L
L2
AIR INTAKE
1-15-1-T
TH
HVAC/DC
1-12-3-Q
QF
DECK SHOWER
1-11-1-L
LW

MAIN DECK



PILOTHOUSE